MACHINE LEARNING REGRESSION FOR ESTIMATING CHARACTERISTICS OF LOW-THRUST TRANSFERS

A Thesis
Presented to
The Academic Faculty

By

Gene L. Chen

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Aerospace Engineering

Georgia Institute of Technology

May 2019

MACHINE LEARNING REGRESSION FOR ESTIMATING CHARACTERISTICS OF LOW-THRUST TRANSFERS

Approved by:

Dr. Dimitri Mavris, Advisor
Guggenheim School of Aerospace Engineering
Georgia Institute of Technology

Dr. Alicia Sudol
Guggenheim School of Aerospace Engineering
Georgia Institute of Technology

Dr. Michael Steffens
Guggenheim School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: April 15, 2019

To my parents, thanks for the support.

ACKNOWLEDGEMENTS

Firstly, I would like to thank Dr. Dimitri Mavris for giving me the opportunity to pursue a Master's degree at the Aerospace Systems Design Laboratory. It has been quite the experience. Also, many thanks to my committee members — Dr. Alicia Sudol and Dr. Michael Steffens — for taking the time to review my work and give suggestions on how to improve - it means a lot to me. Additionally, thanks to Dr. Patel and Dr. Antony for helping me understand their work in trajectory optimization.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Compared to chemical propulsion, electric propulsion is much more efficient in its use of propellant in space. An electric propulsion engine can easily provide an order of magnitude more velocity imparted, while using the same propellant mass. This advantage is applied by space mission designers: enabling greater payload mass delivered or mission flexibility. However, there are significant drawbacks for electric propulsion. Firstly, the achieved thrust is minuscule, which rules out its application in launch vehicles, since the force of gravity cannot be counteracted in nearly any situation. Also, the computational difficulty of finding a suitable low-thrust trajectory is much more than that of finding a trajectory where the thrust can be assumed to be applied instantaneously, which can be a reasonable assumption in missions involving chemical propulsion. This is due to the long thrusting durations typical of electric propulsion missions, requiring a computer program to generate and store spacecraft states and controls at many time points along the trajectory. The high computation time demanded by a low-thrust trajectory solver has the greatest impact on problems involving large search spaces, where there exists a large number of potential targets to visit, and the additional decision of timing for starting the maneuver burn. The infeasibility of calculating high-fidelity trajectory solutions to all transfer combinations would typically impact planning for missions to space debris or small Solar System bodies, where potential destinations number in the thousands. The goal of this work was to use machine learning techniques to create an accurate and computationally efficient manner of estimating the fuel and time cost of a transfer between a wide variety of orbits. A particular focus was placed on flyby-type problems, where the velocity at the target is unconstrained. This is of interest because little literature has investigated flyby transfers to targets that are of completely disparate orbit types. Existing literature tends to focus on transfers between objects that at least have a common orbital inclination, even if the longitude of ascending nodes may differ (e.g. 9^{th} Global Trajectory Optimization Competition (GTOC)). In

this thesis, sequential design choices were made for the solver used to create the training data. It was found that using the Sims-Flanagan method, a direct method, was the most practical way of finding valid trajectories for creating the training data. Several machine learning methods were used to predict fuel and time expenditures of the trajectories - support vector regression and gradient boosting were among the highest-scoring techniques, but no one machine learning technique greatly outperformed any other. Thus, this thesis demonstrates that machine learning methods can produce quick and accurate-enough (for sequence generation problems, potentially) predictions of fuel and time expenditures of general low-thrust transfers.

CHAPTER 1
INTRODUCTION AND MOTIVATION

Trajectory planning is a time-consuming and difficult part of spacecraft mission design. There are often requirements for a spacecraft to visit specific points in space, with tight fuel margins. Therein lies the difficulty, as it is trivial to create an infeasible trajectory, but difficult to create an optimal one. Additionally, the ability for a spacecraft to visit multiple destinations is often desired by mission planners; however, this comes at a cost of making the trajectory planning problem much harder. This is because there is additional complexity in choosing not only the destinations the spacecraft would visit, but also the order in which the spacecraft visits them. At the same time, trajectories must be found between candidate destinations. Since the number of destinations can easily reach the tens of thousands and even climb, as with the number of known asteroids near the Earth, seen in Figure 1.1, a large dimensionality can be anticipated from the problem. Likewise, the European Space Agency (ESA) predicts that there are around 29,000 pieces of space debris larger than 10 cm in size. Further compounding the problem is the fact that the mass, power, and other properties of the spacecraft could change as it travels along its trajectory[1]. The extent of the difficulty of sequential trajectory planning is best exemplified in the Global Trajectory Optimization Competition (GTOC), an international competition, occurring every 1-2 years, to create the most optimal trajectory for the given problem. As put forth by Izzo [3], the trajectory optimization problem can be interpreted as a global optimization task. In general, the global optimization problems in GTOC are designed to be complex, and the GTOC 1 problem achieved this complexity by allowing for sequences of planetary flybys and having a long launch window[3]. In practice, mission planners use sequences of flybys of objects for the gravity assists they offer (saving fuel en route to another target) or to visit multiple destinations with a single spacecraft. For the latter purpose, there is a significant

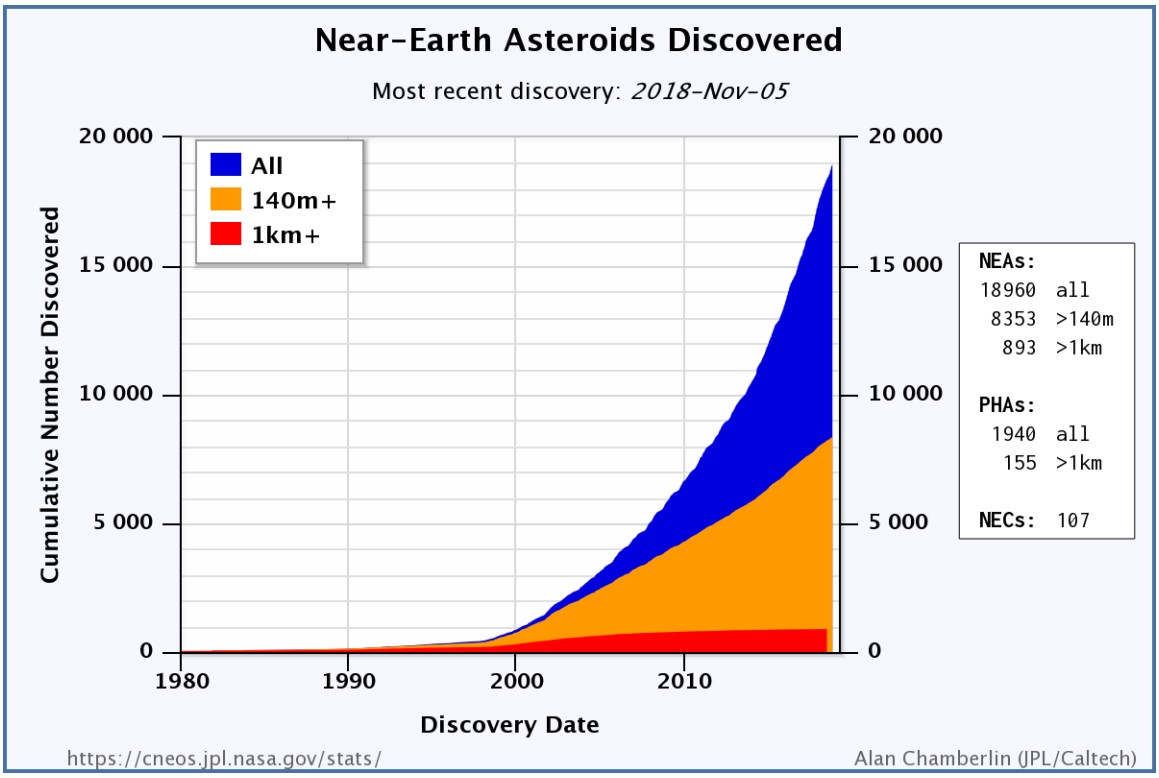
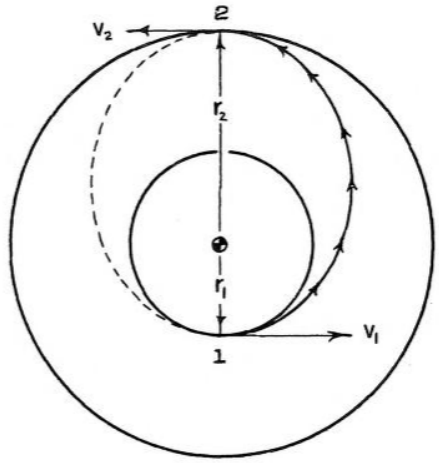


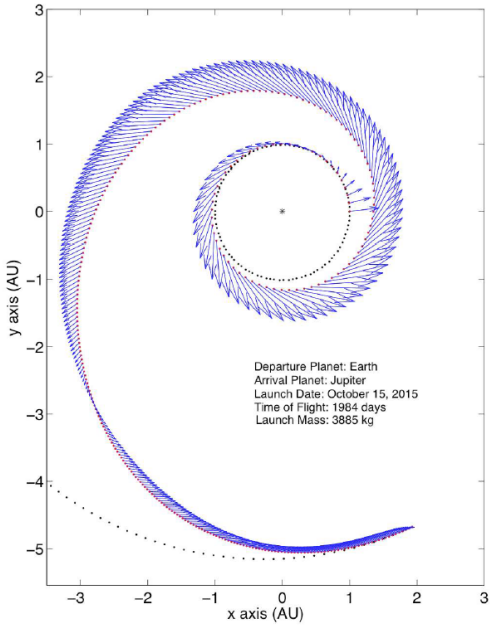
Figure 1.1: Cumulative number of Near-Earth Asteroids (NEAs) Discovered in Recent Years. (From [2])

advantage in returned scientific or reconnaissance data if a spacecraft can visit multiple asteroids or space debris, respectively. For example, the *Dawn* spacecraft, launched in 2007, was the first spacecraft to orbit two main-body asteroids, which allowed for detailed maps to be made of both bodies. Typically, traveling from one destination to another in space requires a maneuver, or change in velocity, in the spacecraft. Electric propulsion (EP) is an efficient way of imparting such a change in velocity. Such efficiency is often measured in terms of I_{sp}, or specific impulse. The gulf in efficiency of the more traditional chemical rocket propulsion and EP (also referred to as low-thrust or continuous thrust systems) is abundantly clear when comparing their typical specific impulses, as chemical rocket engines tend to have specific impulses of less than 500 seconds, while it is not unusual for specific impulses of EP engines to exceed 4000 seconds. However, the main drawback of EP thrusters is that many of them have maximum thrusts on the order of millinewtons, and require large amounts of electricity. For example, HiPEP, the electric propulsion engine

developed at NASA, creates 600 mN of thrust with a specific impulse of 9150 seconds, while consuming 34.6 kW of power[4]. In the field of trajectory planning, this disadvantage manifests in the form of computational difficulty, as the impulsive thrust assumption commonly used for chemical propulsion missions can no longer be used. This forces the low-thrust optimization program to solve a continuous optimization problem rather than a comparatively-simpler discrete optimization problem that is associated with impulsive trajectory optimization[5]. Figure 1.2 visualizes the difference between impulsive thrust and continuous thrust trajectories. Trajectory simulations involving only unperturbed, impulsive maneuvers can assume that the spacecraft follows conic sections between maneuvers. This is an invalid assumption for low-thrust maneuvers since the trajectory shape is always changing while the spacecraft is thrusting. In most cases, a low-thrust spacecraft is also thrusting for a large portion of the time-of-flight from the start to the end of the trajectory. The computational difficulty of solving a GTOC-class problem is directly correlated with



(a) A Hohmann transfer, showing two impulsive maneuvers. (From [6])



(b) A continuous-thrust transfer. Blue arrows indicate thrust direction and magnitude. (From [7])

Figure 1.2: Comparison between a notional impulsive and a continuous-thrust trajectory from Earth to Jupiter.

the computational difficulty of solving a transfer within that problem. This, along with the relative difficulty of low-thrust transfer computation, leads to an incongruity where it can be a trivial matter to create a trajectory for sequence of targets using impulsive maneuvers, but exceedingly difficult and time-consuming to create a low-thrust trajectory for the same sequence of targets. It is therefore advantageous to have a methodology for quickly predicting the fuel and time costs for a general low-thrust transfer, which could enable more optimal choices of destination for each step in a sequence of targets to visit. This is because current sequence generation methods, such as beam search[8] and genetic algorithms[9], rely on heuristics in order to make decisions for which bodies to visit, and which bodies to not consider. These heuristics are required because of the large computational cost of finding low-thrust transfers for all bodies in the list of potential targets. A machine learning predictor would preclude the need for these potentially-inaccurate heuristics, and all targets could be evaluated for their suitability in terms of transfer cost.

1.1 Thesis Overview

This chapter began with an introduction to the problem of planning low-thrust orbit transfers. In the next chapter, the research objective and research questions are presented, along with a literature review to provide additional details on the ways the problem has been traditionally solved, with emphasis on the advantages and disadvantages of various methods. Chapter 3 outlines the approach to solve the problem, Chapter 4 details the experiments that were done to evaluate the research hypotheses. The final chapter includes an overview of the contributions to the field, and ends with remarks on potential avenues of further research and enhancements to the research presented.

CHAPTER 2
BACKGROUND

The high computational effort required to create low-thrust trajectories presents a problem to mission designers. Creating one trajectory is relatively quick - one low-fidelity trajectory takes about 30 seconds to converge, based on the author's previous work. Taking into account the large number of potential destinations, the problem quickly becomes intractable, because of the many choices of destination at every step in the sequence of destinations to visit. In previous research, some methods have been used to reduce the computations needed to find the relevant parameters of the low thrust trajectory to aid mission designers. These methods often estimate the suitability of a particular transfer so that the full trajectory control solution does not need to be found. One way low-thrust trajectories have been estimated is by using a Lambert solver[10]. Sometimes this is an initial, first-order approximation to be further refined[11]. A Lambert solver uses the solution of the Lambert problem, which seeks to find a Keplerian orbit that fits two positions and a time of flight. The two positions in this instance would be a point on the initial and final orbits. The results of a Lambert solver are usually represented in porkchop plots, which display the ΔV required for a transfer, with arrival time and departure time as axes. An example of a porkchop plot depicting a transfer between Earth and an asteroid can be seen in Figure 2.1, and one for the Earth-Mars transfer can be seen in Figure 2.2. While computationally cheap and intuitive, this is a very inaccurate way of estimating low-thrust trajectories, as seen in Figure 2.3. A method used in recent literature are phasing indicators[14][15]. The inputs are usually the state and orbital parameters of the initial and destination orbits, and it outputs a value correlated with the ΔV required to reach destination orbit. Much like the Lambert solver, the phasing indicator is only useful in conjunction with a global optimization method for building sequences, since it is only concerned with creating a quick

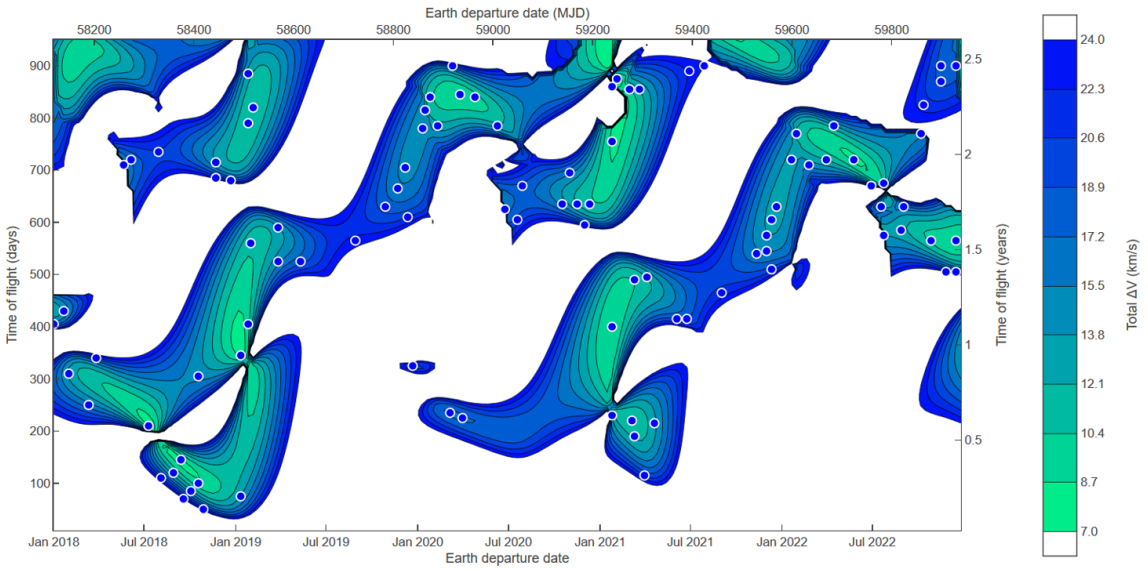


Figure 2.1: An example of a porkchop plot. Colors represent the ΔV for the transfer at the corresponding departure time and time-of-flight. (From [12])

ranking of transfers. Thus the utility of phasing indicators is limited, since the fuel and time expenditure of the transfer are not given. Lastly, these indicators are not necessarily suitable for targets in the Low Earth Orbit (LEO) orbit regime, where satellites and debris in high inclinations and eccentricities are common. In the asteroid belt, which is a target set investigated in instances of previous literature that incorporated phasing indicators, there are very few asteroids of high inclination or eccentricities, so the utility of phasing indicators was not demonstrated for transfers between asteroids of unusual orbital characteristics. Also, phasing indicators were only used to rank rendezvous-type transfers, not flybys, which limits the mission types these indicators can be used for. Due to the large effort required to create a low thrust trajectory, machine learning methods have also been applied to the problem. This is especially fitting as machine learning is appropriate for highly nonlinear problems. Since machine learning also being highly adaptable to a variety of problems, it has been applied to many problems relating to the low-thrust trajectory optimization in previous literature: creating a state and control from time t_i to t_{i+1} as part of a collocation method[16], regression for initial mass required for an optimal transfer between asteroids[10], neurocontrol of a low-thrust, multiphase trajectory[17], among others.

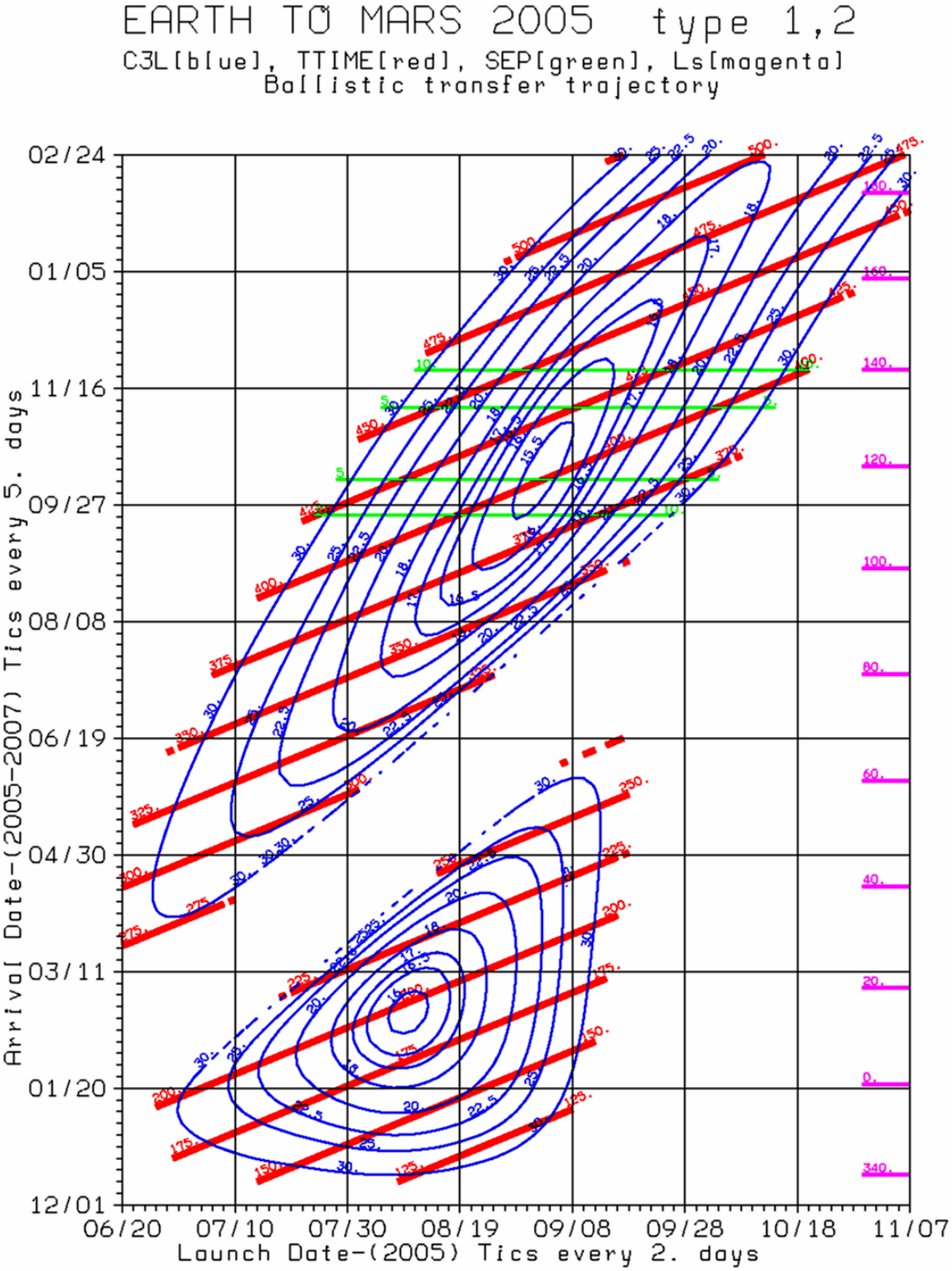


Figure 2.2: Porkchop plot for Earth-Mars transfers in 2005. Blue contours indicate equal ΔV transfers, and red lines indicate transfers with equal time-of-flight (From [13])

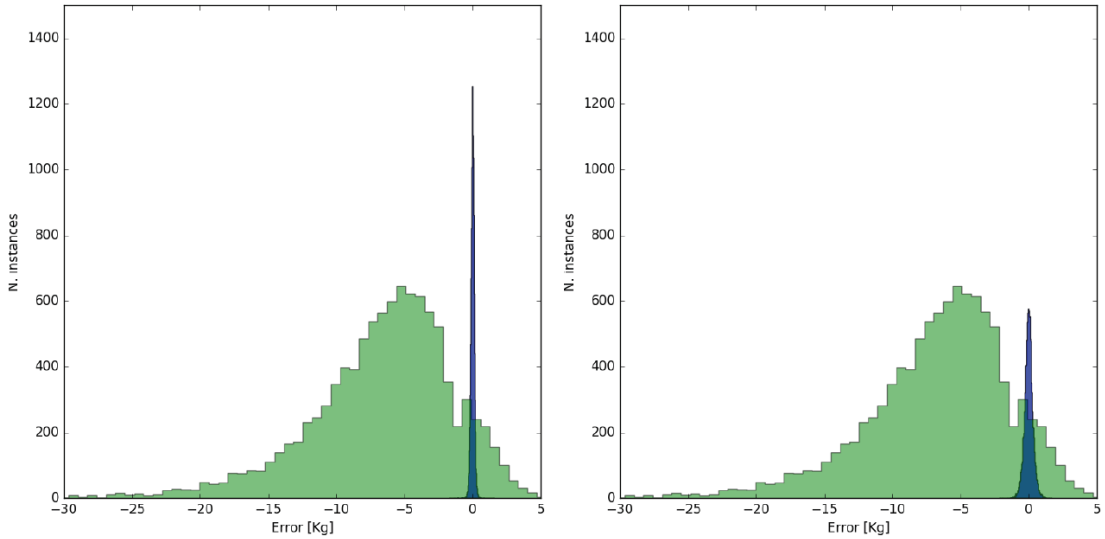


Figure 2.3: Distribution of prediction errors for two machine learning regressors (*blue*) and a Lambert solver (*green*). (From [10])

However, previous research has been lacking in the area of orbit generality, as both [10] and [17] only simulate transfers between two nearly-coplanar orbits. Coplanar orbit transfers are only a small subset of possible orbit transfers.

Research Objective: To create a predictive model using machine learning that quickly and accurately predicts the time and fuel costs associated with a general continuous-thrust orbit transfer.

More specifically, this predictive model needed to predict, with comparable accuracy to current machine learning implementations, the fuel and time cost of transfers of a much larger variation in the differences of orbital elements between initial and the final states. Additionally, this needed to be done with a limited training data set, or the computational cost of creating the training data could become comparable to brute-forcing a trajectory solution by solving all possible sequences of targets, especially for a short target list. In order to conduct machine learning, a large amount of training data was created. This is even more critical for the purposes of this thesis - because of the large variety of orbit transfers that will be solved, a very large number of transfers must be included in the training data set in order to prevent a future input from being far outside the training data set,

which may cause increased prediction error. In order to validate the resulting regressor, additional data must be created and tested. To give an idea of the scale involved, in [10], the researchers used 50,000 optimal trajectories as the training set, and 10,000 in the validation set. Without parallelization, solving all 60,000 optimal trajectories could take over 20 days of wall time for a standard desktop computer, assuming 30 seconds per trajectory. Due to limited computing resources, design of experiments techniques were used to minimize the number of runs to be solved. Because a large combinatorial space is being examined, the method of creating training data must not only be accurate and optimal, but also fast. Most importantly, the trajectory optimization method must reliably work without suffering from significant nonconvergence issues or other such interruptions. Optimality of the training data is also a critical aspect since the results of the machine learning regressor will only be as optimal as its training data, and a lack of optimality may preclude the use of the machine learning regressor for certain applications, such as in sequences of orbital maneuvers. Olympio [18] mentions that in GTOC4, teams found it difficult to have long sequences of transfers converge while maintaining optimality because even a small increase in thrust acceleration (due to expenditure of fuel) can cause an infeasible trajectory solution to become feasible. An impossible transfer early in the mission can become possible in a later phase. Direct application of the machine learning regressor to create multiphase trajectories leveraging global optimization techniques such as tree search algorithms and genetic algorithms was not the intention of the work in this thesis. This is acceptable since the number of times full thrust control histories are generated during a notional mission design phase should be relatively low and thus do not suffer as much from inefficiency. Instead, a would-be mission designer could use the procedure in this thesis to reduce the size of the search space. The low-thrust trajectory generation problem is typically solved in one of two ways: direct methods and indirect methods. These two overarching methods have their own advantages and disadvantages. Direct methods[19], as its name implies, directly solve for the physical variables in the low-thrust problem. They reframe the prob-

lem into a nonlinear programming problem and either discretize the trajectory into many arcs divided by small impulsive-thrust maneuvers[20] or by parameterizing the control using a set of functions[21]. Although direct methods are often less computationally intensive, they suffer from lack of optimality if the basis functions are not correctly selected. Also, some direct methods have difficulty incorporating the effects of perturbations such as third-body gravity, drag, and solar radiation pressure. Indirect methods use calculus of variations with Pontryagin's Maximum Principle to turn the trajectory problem into a two-point boundary value problem. Indirect methods tend to produce very near-optimal results, but are extremely sensitive to initial guesses of the costate variables, which can also be non-intuitive[20]. Also, indirect solvers are usually not problem-agnostic, as the costate equations, which describe the relationships between the costate and state variables, must be defined before solving the problem. Some newer approaches, such as Quasilinear Chebyshev-Picard Iteration method (QCPI) specifically address this shortcoming, but may still require some degree of problem-specific tuning[22]. Some of the disadvantages of indirect methods can be mitigated by using a solution produced by a direct method as an initial guess for the states and/or costates for the indirect method. Even though this technique would further increase the already long computation times of indirect methods, it may be required for the indirect method to even converge. Therefore indirect methods are usually limited to cases where higher fidelity is needed, the problem is well-defined, and a high optimality is desired. To fulfill the research objective, the first research question that must be asked pertains to the choice between using a direct or indirect solver:

Research Question 1: Would a direct or indirect method be more suitable for creating the training data?

Due to the fact that indirect methods can be faster than direct methods when a good initial guess is provided[18][23], the hypothesis for **Research Question 1** is that an indirect method would be better suited for creating the training data. Prior implementations of machine learning on the low-thrust fuel and time cost prediction problem tended to focus on

transfers to and between NEAs[10][24]. By definition, NEAs are asteroids that have a periapsis within 1.3 AU (AU standing for Astronomical Unit, or the average distance between the Earth and the Sun)[25]. As a result, this is a constrained set of targets when compared to the set of all known asteroids. Similarly, the range of orbital elements is also constrained, as the limit on the largeness of the periapsis distance impacts both the semimajor axis and the eccentricity. This is in addition to the fact that NEAs typically have low orbit inclinations. Indeed, the Hungaria group of asteroids, commonly described as a high-inclination group[26], has a relatively (when compared to, say, satellites in Earth orbit) modest inclination of 16-35°. In this thesis, transfers to a wider range of targets are considered, so the following question must be asked:

Research Question 2: Do machine learning algorithms maintain accuracy for a larger design space of transfers?

Machine learning algorithms have been used to predict the ΔV expenditure to visit NEA targets. This was created in response to the fact that the phasing indicators are limited in accuracy for longer hops between asteroids. In general, machine learning regression methods learn the relationship between inputs and outputs by adjusting a model using training data[27]. Several machine learning regression algorithms have been implemented for the low-thrust trajectories and other applications. All of the involved methods are supervised learning methods, since a particular predicted output is needed. There are two types of supervised learning: classification and regression. On the other hand, unsupervised learning is a way to cluster the input data in various groups to better understand the distribution of data or to visualize the data in a lower-dimensional space[28]. Supervised regression is used since the goal of this thesis is to demonstrate a prediction capability for continuous output variables, namely, the fuel and time-of-flight cost of a flyby maneuver. Supervised classification between unfeasible and feasible trajectories was not sought since an unfeasible trajectory was understood by the author to be equivalent to an excessively expensive maneuver. The following supervised learning regression methods were used in this thesis:

- **Decision tree**: Decision trees represent a segmentation of data that is created by applying a series of rules[29]. The main benefits of using this method is that the relative importance of predictors is easily extracted. However, this method is inferior for nonlinear data, which is anticipated in this particular problem. Decision trees are more suited for categorization problems rather than regression, as is implied in the name. The structure of a three-layer decision tree can be seen in Figure 2.4.

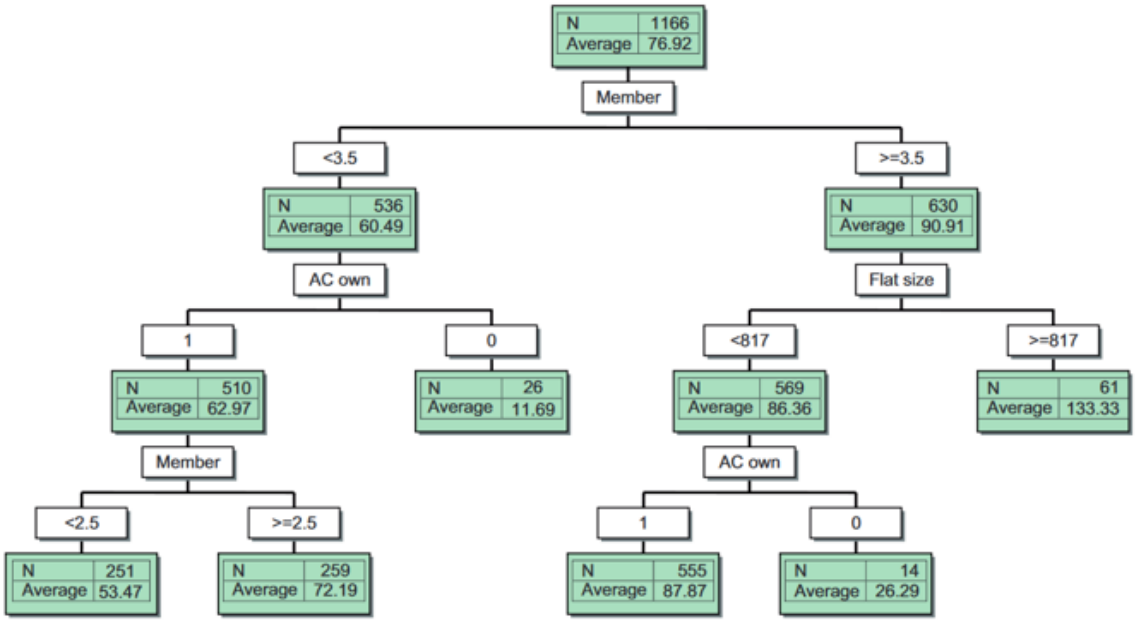


Figure 2.4: Decision Tree. (From [29])

- **Random forest**: This is a modified version of bagged trees. Bagged trees is a method that takes some bootstrap subset from the training data set and gradually building fitted trees from there, and finally averaging all the bagged trees[30]. In the random forest method, a random subset of predictors is chosen from each split. This has the advantage of reducing the variance of the prediction error. The basic workflow of a random forest can be seen in Figure 2.5.

- **Gradient boosting**: This is an approximation technique that uses the steepest descent method to forward stagewise estimation. This method decreases the amount of

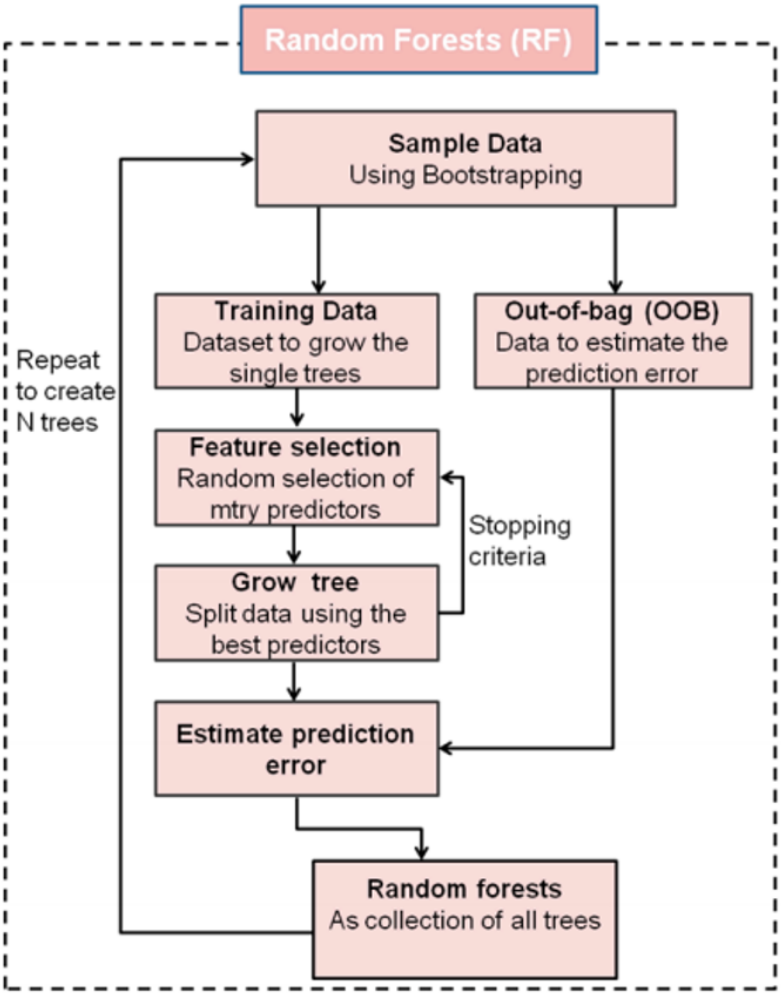


Figure 2.5: Random Forest. (From [31])

squared-error loss and exponential loss resulting from non-robust data. A flowchart of the gradient boosting method can be seen in Figure 2.6.

- **Neural network**: This is a structure of artificial neurons connected and arranged in layers. There is one neuron for each input in the input layer and one neuron for each output in the output layer. Any number of neurons can exist in the intermediate layers, but an increased number of layers generally increases the chance of decreased performance due to overfitting[24]. Other choices such as initialization of weights, shape of the nonlinearity, and learning rate can affect the performance of the neural network[27]. The workflow of a neural network can be seen in Figure 2.7.

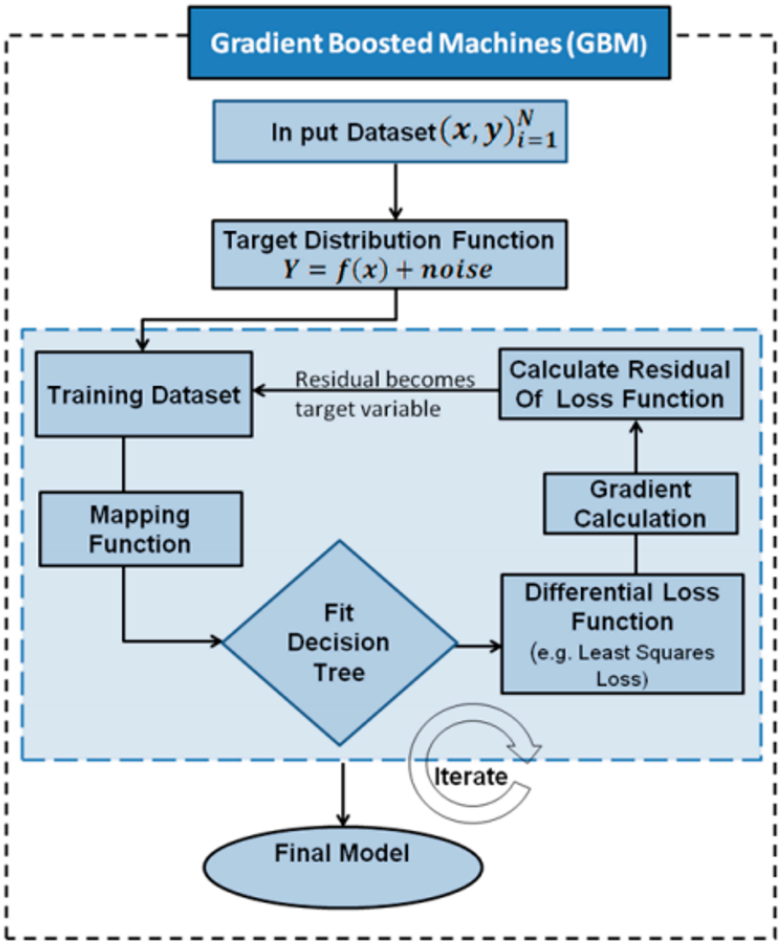


Figure 2.6: Gradient Boosting. (From [31])

- **Support vector regression**: This method creates a linear model over the mapped samples to a higher dimensional space. The higher dimensional space itself is non-linearly related to the original inputs. The standard variant of support vector regression utilizes Vapnik's ϵ-insensitive cost function. The workflow and structure of the support vector regression method can be seen in Figure 2.8.

- **Kernel ridge regression**: This method seeks to minimize the residuals in the higher dimensional space as described in support vector regression. It is essentially a kernel version of the regularized least squares linear regression. There is an advantage in the kernel ridge regression that only the regularization parameter and the kernel parameters need to be tuned, and that the closed-form solution can be expressed, so

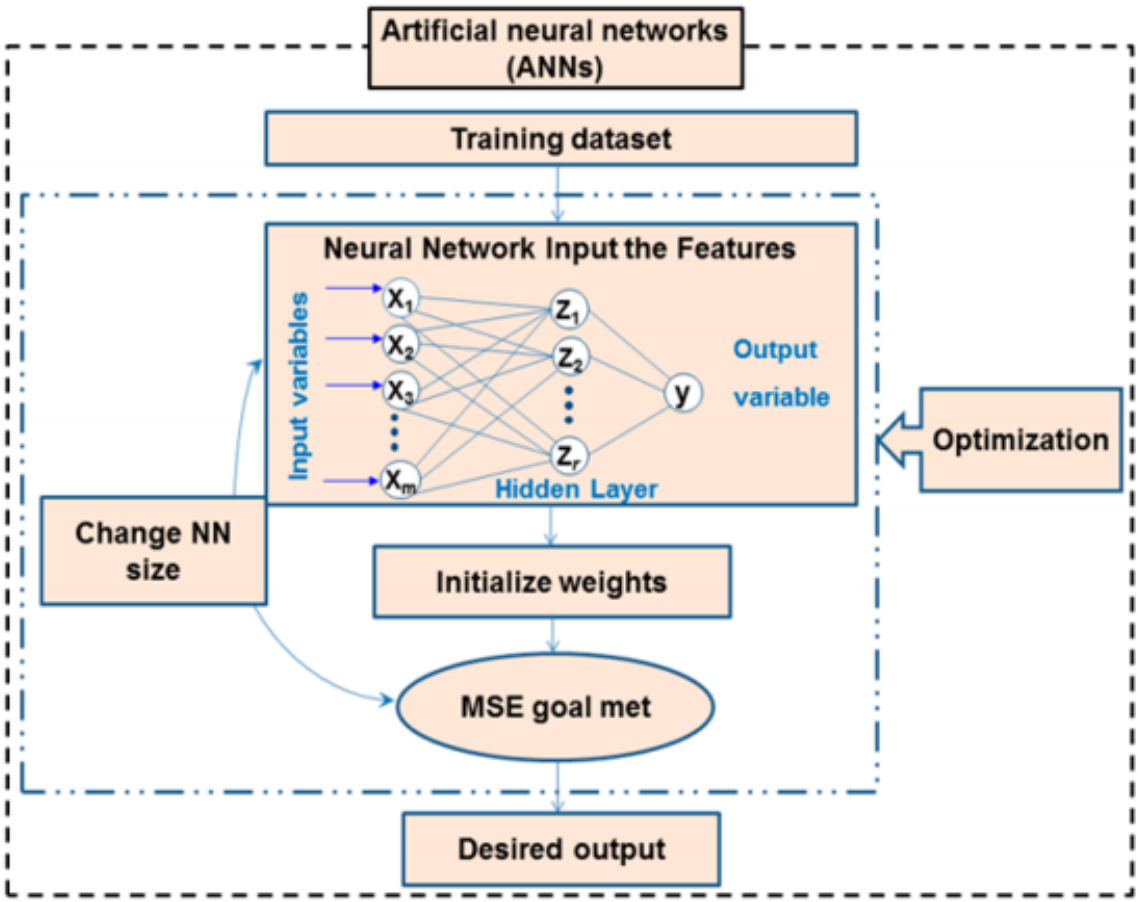


Figure 2.7: Artificial neural network. (From [31])

quadratic programming is not needed. The main disadvantage is that the kernel ridge regression model is not sparse, and that all training data are incorporated in the found solution.

Due to its performance in previous literature[10], the gradient boosting method was hypothesized to be the most accurate machine learning method tested in this thesis.

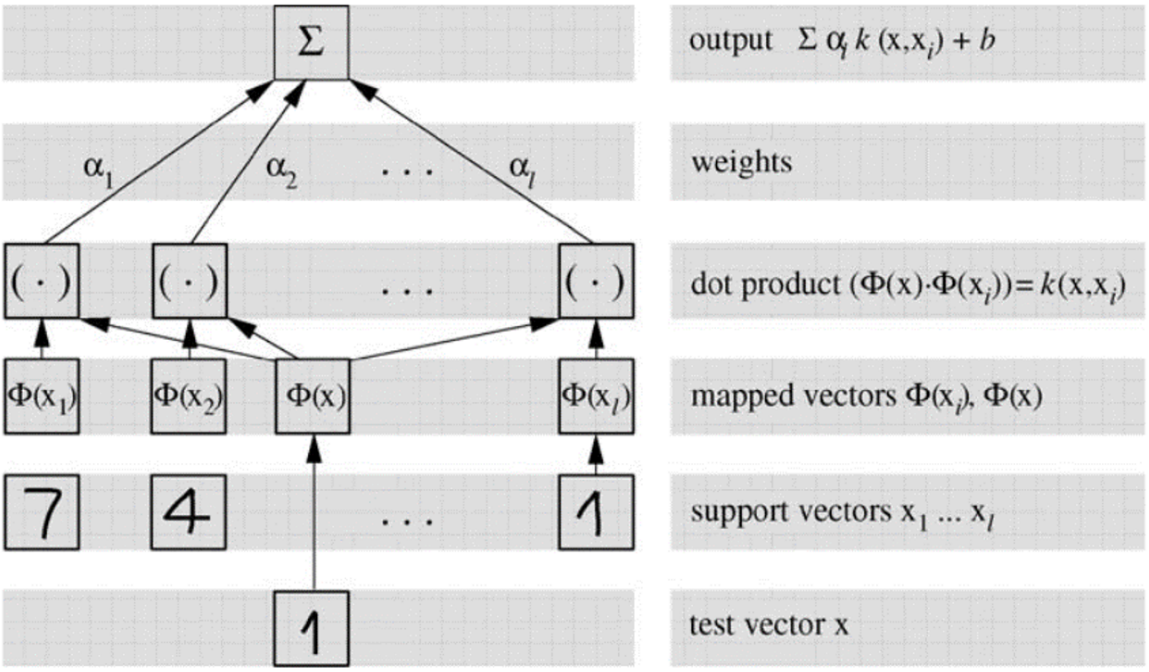


Figure 2.8: Support vector regression process. (From [32])

CHAPTER 3
APPROACH

The chapter describes the testing scenario and methods used to generate low-thrust trajectories.

3.1 Scenario Definition

To get an adequate spread of the possible orbit transfers the heliocentric orbit regime, the initial and final orbital elements are varied so that the semimajor axis is on a range of 0.6 AU $< a <$ 3 AU, an eccentricity, $e <$ 0.6, and inclination, $i < 90°$(to ensure prograde orbits), with argument of the perigee (ω), right ascension of the ascending node (Ω), and mean anomaly (M) distributed throughout 0 to 360°. For reference, an image showing the physical meaning of the first 5 classical orbital elements is shown in Figure 3.1 - the last orbital element, M, or the mean anomaly, describes where on the orbit an object is at a reference point in time. Transfer times were effectively limited to around 11 years, which helped decrease the run time since a longer trajectory would cause a longer run time to calculate due to the need for more trajectory segments. This consideration also factored into the upper limit for semimajor axes in this analysis, since a smaller semimajor axis causes a smaller orbit and thus a shorter trajectory. Examples of transfers within this design space of orbital elements include transfers between Hungaria asteroids and Ceres, and Earth-Mars transfers. The limits on the orbital parameters are such that only prograde orbits are considered, with circular to moderately elliptical orbit shapes. The semimajor axis limits correspond to transfers involving bodies ranging from being closer to the Sun than Venus, and out beyond the orbit of Ceres. The parameters of the spacecraft used for the training data are kept constant in order to save on computation time, since the orbit transfers involved, as opposed to spacecraft sizing, are the focus of this research

effort. The simulated spacecraft in this research will have an initial mass of 3000 kg and a notional electric propulsion engine with a maximum thrust of 0.2 N at a constant specific impulse of 4000 seconds. These specifications are realistic since they are close to the NEXT (NASA Evolutionary Xenon Thruster) engine from Glenn Research Center, which has a maximum thrust of 0.237 N and a specific impulse of 4170 seconds[33]. The assumption of constant specific impulse was also used in the creation of the training data. The mass of the simulated spacecraft is in the ballpark of Dawn and other deep space spacecraft.

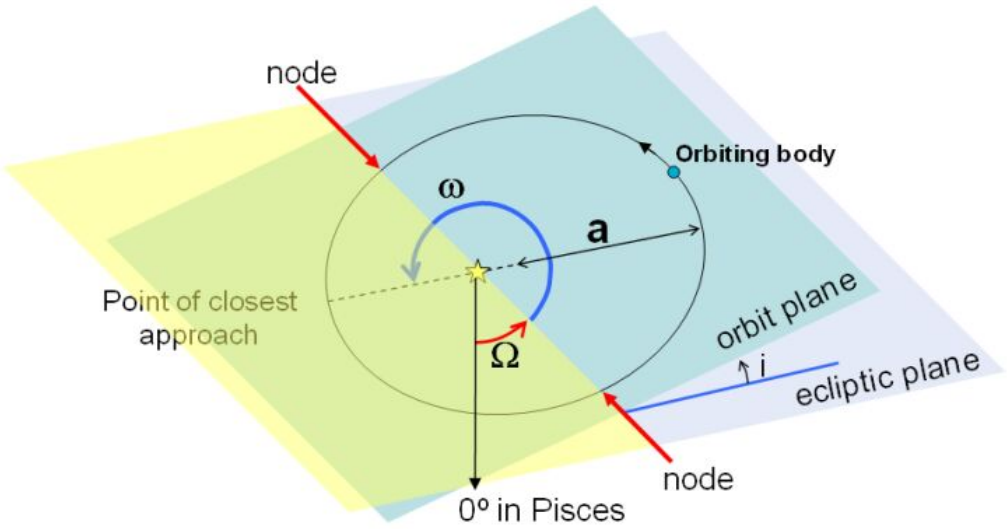


Figure 3.1: Physical Meaning of Orbital Elements (from [34])

3.2 Spacecraft Dynamics

In the two-body problem, the dynamics for a body undergoing only gravitational forces and a thrust acceleration can be expressed as

$$\ddot{\boldsymbol{r}} = -\frac{\mu \boldsymbol{r}}{|\boldsymbol{r}|^3} + \frac{\boldsymbol{T}}{m} \tag{3.1}$$

where $\boldsymbol{r}$ is the position vector of the spacecraft in an inertial frame, μ is the gravitational parameter of the central body, $\boldsymbol{T}$ represents the thrust vector, and m is the mass of the

spacecraft. The relationship between the mass and the thrust can be defined as such

$$\dot{m} = -\frac{|\boldsymbol{T}|}{g_0 I_{sp}} \tag{3.2}$$

3.3 Choice Between Direct and Indirect Method

The hypothesis that an indirect method would be more effective for creating the training data largely relied on the ability to create good initial guesses with trivial computation time. However, this was not possible due to several factors. So-called initial guesses were feasible trajectories in their own right. More importantly, the computation time for these methods were far from trivial - depending on the input parameters, creating trajectories took between 20 seconds and upwards of 3 minutes per trajectory. Using an indirect method would only add to the computation time. Also, less than half of indirect runs resulted in feasible trajectories, which would decrease the amount of training data available if an indirect method was selected to create the training data. Lack of adequate quantity of training data was deemed more critical than the lack of optimality in the training data. Because of these reasons, it was decided that a direct method would be more suitable for generating the training data, leading to another research question:

Research Question 1a: What type of direct method should be used to create the training data?

Two direct method solvers were investigated for suitability in producing feasible trajectories: the Chebyshev polynomial method and the Sims-Flanagan method.

3.3.1 Chebyshev polynomial method

The highlights of the procedure in [35] are repeated here, and are based on [36]. In a trajectory, the Chebyshev polynomials can represent the position coordinate *w* as

$$w(t) = \sum_{j=0}^{N-1} c_{w,j} T_j(t) \tag{3.3}$$

and a generic velocity coordinate $\dot{w}$ can be represented as

$$\dot{w}(t) = \sum_{j=0}^{N-1} c_{w,j} \dot{T}_j(t) \tag{3.4}$$

where $c_{w,j}$ are a path coefficients and T_j is the j^{th} order Chebyshev polynomial. The Chebyshev polynomials can be found found using the following equation:

$$T_j(t) = \begin{cases} 1 & \text{if } j = 0 \\ \tau & \text{if } j = 1 \\ 2\tau T_{j-1}(\tau) - T_{j-2}(\tau) & \text{if } j \geq 2 \end{cases} \tag{3.5}$$

and $\dot{T}_j(t)$ can be found by taking the derivative of (3.5) with respect to τ. Note that τ is the transformed time variable, which relates to the untransformed time variable by the following equations

$$\tau = 2\frac{t - t_0}{t_f - t_0} - 1 \tag{3.6}$$

$$\dot{\tau} = \frac{2}{t_f - t_0} \tag{3.7}$$

where t_0 marks the beginning of the transfer, and t_f the end. At this point, the trajectory is most likely still infeasible and requires additional refinement by applying either the generalized Newton method to the problem's constraints and controls, or the multiple-shooting method.

3.3.2 Sims-Flanagan method

The Sims-Flanagan method was initially proposed in 1997 by Jon Sims and Steve Flanagan[20]. The method works by splitting a low-thrust trajectory into segments. Between two segments exists a match point where the mismatches in position, velocity, and mass must be within a given tolerance. Typically, each segment would also simulate a small impulsive maneuver at the midpoint of each segment. The structure of the match points and segments can be seen in Figure 3.2.The figure shows a trajectory with three destinations visited, whereas in this thesis, only a starting orbit and ending orbit are defined. The Sims-Flanagan method is notable for its low computation cost, ease of implementation, and large radius of convergence[37]. However, the problem dimensionality of the Sims-Flanagan method is directly proportional to the number of segments used. Also, this is a low-fidelity solver as it does not account for perturbations unless the segments are propagated numerically and thrust calculated continuously. If continuous thrust is used, the computational advantage of the Sims-Flanagan method is largely negated (with respect to other direct solvers) unless reclaimed by using adaptive time meshing techniques, such as through Sundman transformation[19][38]. The package PyKEP does allow use of both continuous thrust calculation and adaptive time meshing together with the Sims-Flanagan method, but this was ultimately not used for the training data in order to minimize computation time and maximize the number of trajectories solved, to better train the machine learning algorithms. For all Sims-Flanagan trajectories found and used in the machine learning algorithm training and validation, the segment bounds were equally spaced in time. Algorithmically, PyKEP implements the Sims-Flanagan method by employing forward-backward propagation. This is assisted by the package Parallel Global Multiobjective Optimizer, or PyGMO[37]. The use of PyGMO allows multiple candidate solutions for a transfer to be investigated simultaneously. As mentioned previously, the Sims-Flanagan method finds the mismatches at each match point and attempts to minimize them. Specifically, the parameters of importance are the mismatches in position, velocity, and mass. There is a certain amount of velocity and

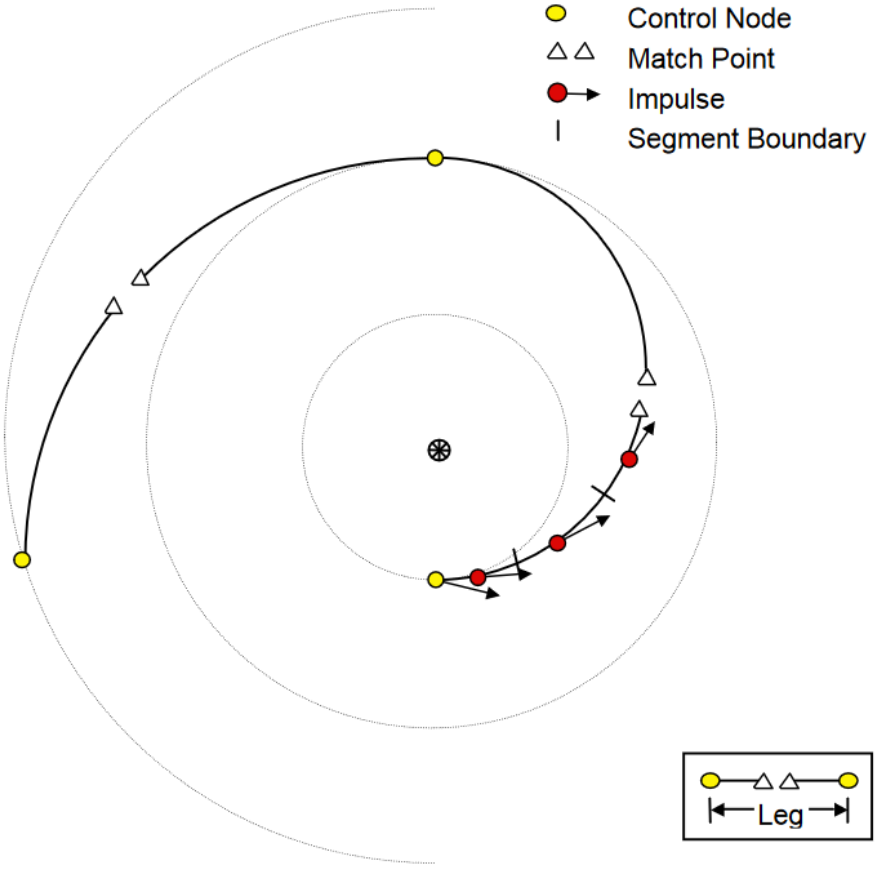


Figure 3.2: Basics of the Sims-Flanagan Method (from [39])

mass mismatch that is allowed in addition to the tolerance level, given by the limits of the propulsion system performance on the leg:

$$\Delta V_{max} = (\frac{T_{max}}{m})(t_f - t_0) \tag{3.8}$$

where T_{max} is the maximum thrust of the EP engine and t_0 marks the beginning of the leg, and t_f the end. Similarly, the mass is allowed to mismatch via the rocket equation:

$$m_{i+1} = m_i \exp{(\frac{-\Delta V_i}{g_0 I_{sp}})} \tag{3.9}$$

where m_i and ΔV_i are the mass and change in velocity at the *i*-th segment, respectively. The minimization of mismatches occurs as PyKEP transcribes the Sims-Flanagan problem

into a nonlinear programming problem (NLP). SNOPT7 is used to solve the NLP, and as a result, much of the computation time involved is in running SNOPT7. The decision vector passed to the NLP consists of the departure time, the departure velocity relative to the initial object, the time-of-flight, the arrival velocity relative to the destination object, the arrival mass, as well as the velocity impulse at each leg. SNOPT7's task would be to find the components of an impulsive thrust at each segment of the trajectory such that the mismatch constraints are met, while minimizing the thrust magnitude. As originally formulated by Sims and Flanagan, the propagation works by using the state-transition matrix Φ_{x_i} to propagate forwards from the starting point through $n_{forwards}$ segments and backwards from the arrival point through $n_{backwards}$ segments.

$$x_{i+1}^{m^-} = \Phi_{x_i}(\frac{\Delta t}{2})x_i \tag{3.10a}$$

$$x_{i+1}^{m^+} = x_{i+1}^{m^-} + \Delta x_{i+1} \tag{3.10b}$$

$$x_{i+1} = \Phi_{x_{i+1}^{m^+}}(\frac{\Delta t}{2})x_{i+1}^{m^+} \tag{3.10c}$$

where the Δx_{i+1} is the vector describing the change in state at the midpoint of the segment:

$$\Delta x_{i+1} = [0, 0, 0, \Delta v_x, \Delta v_y, \Delta v_z, \Delta m] \tag{3.11}$$

which contains the commanded impulsive thrust components and the associated change in mass of the spacecraft. The backwards propagation works in a similar manner,

$$\widetilde{x}_{i+1}^{m^-} = \Phi_{\widetilde{x}_i}(\frac{-\Delta t}{2})\widetilde{x}_i \tag{3.12a}$$

$$\widetilde{x}_{i+1}^{m^+} = \widetilde{x}_{i+1}^{m^-} - \Delta\widetilde{x}_{i+1} \tag{3.12b}$$

$$\widetilde{x}_{i+1} = \Phi_{\widetilde{x}_{i+1}^{m^+}}(\frac{-\Delta t}{2})\widetilde{x}_{i+1}^{m^+} \tag{3.12c}$$

except that the time propagation is done backwards, and the impulsive thrust is subtracted from the state instead of adding to it. A diagram of a 5-segment Sims-Flanagan setup

can be seen in Figure 3.3, with 3 segments in forward propagation and 2 in backwards. PyKEP's implementation of the Sims-Flanagan method is slightly different in that it uses

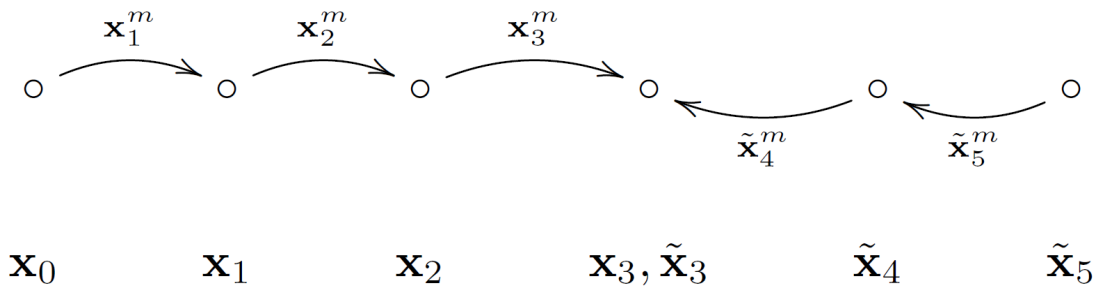


Figure 3.3: Sims-Flanagan Method. In this case, $n_{forwards}$ is 3 and $n_{backwards}$ is 2.

a decision vector of throttles to create the requisite ΔVs at the midpoint of each segment. Also, the high-fidelity mode of the Sims-Flanagan solver in PyKEP can find the trajectory using continuous-thrust segment instead of the state-transition matrix as previously noted. In the high-fidelity mode, the Δx_{i+1} vector is not added to the state at a segment midpoint as with the low-fidelity mode, but is instead incorporated into the propagation. In order to achieve this, Kepler propagation was used. This was observed by [37] to cause a slowdown of a factor 6 to 10; however, the author observed a slowdown of a factor of around 2 to 3. The differences may be attributed to a difference in problem setup - it was observed that for a significant portion of solved trajectories, few trajectory segments were thrusting, which would lower the difference in computation time when comparing high vs. low fidelity settings. The hypothesis for **Research Question 1a** was that the Chebyshev polynomial method would be more suitable for creating the training data. This was supported by literature that named the Chebyshev polynomial method as a way to quickly create feasible trajectories without prior knowledge of the transfer topology, as is the case with shape-based methods[35].

3.4 Monotonic Basin Hopping

In the context of this thesis, monotonic basin hopping was used to enhance the Sims-Flanagan solver. Inclusion of this algorithm greatly increased the likelihood that a feasible solution was found. Monotonic basin hopping is an algorithm first described by David Wales and Jonathan Doye[40]. It works by randomly perturbing an initial input or population of inputs and finding the objective values. Because of this, monotonic basin hopping is inherently a stochastic method and trajectory solutions may not be exactly repeatable. The objective values are checked to see if they are any better than the original input, and if they are, the initial inputs are replaced. The process repeats itself until further improvement cannot be found within a certain iteration limit. This algorithm is especially suitable for problems where many local minima exist, separated by peaks, such as in Figure 3.4. It can be surmised that monotonic basin hopping is appropriate for the orbital transfer problem, as the topography of porkchop plots and low-thrust transfer contours show a familiar pattern of peaks and valleys, as seen in Figure 2.1. Monotonic basin hopping as used in PyGMO

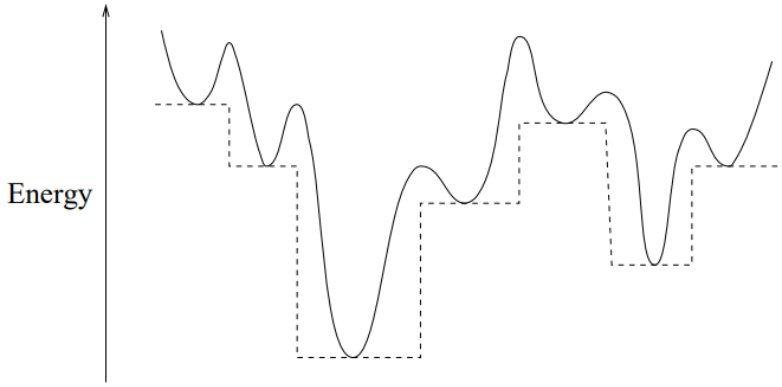


Figure 3.4: Notional objective function topography, to minimize energy (from [40])

allows the user to specify the amount of perturbation as well as the number of consecutive basin hopping iterations that are permitted without an objective value improvement, before the monotonic basin hopping operation is terminated. The pseudocode for monotonic basin hopping as implemented in PyGMO has the structure seen in Algorithm 1:

Algorithm 1: Monotonic basin hopping

```
Initialize random population within bounds;
Evaluate the population;
while iterations without improvement < iteration limit do
    Perturb the population;
    if Best individual is in perturbed population then
        Population = perturbed population;
    else
        iterations without improvement + + ;
    end
end
```

The way individuals in populations are compared is by using PyGMO's default setting, which is to first compare the number of constraints satisfied. If the compared individuals have the same number of constraints satisfied, the norm of the constraint violations is compared. If the individuals have the same constraint violations, the objective value is then determined for the two individuals and the individual that is least-dominated is considered better. After this, if the individuals are still exactly the same, the perturbed population is not considered better than the original population and the count of iterations without improvement increases by one.

3.5 Indirect Optimization

To give a sense of scale to the errors produced by machine learning estimators, a subset of feasible trajectories found via the direct solver were refined using an indirect solver. In order to create the indirect trajectories, Calculus of Variations with Pontryagin's Maximum Principle was used. It is an approach that seeks the optimal control from one state to another. First, the cost, or objective function of a trajectory can be parameterized as

$$J = \phi(\mathbf{x}(t_f), t_f) + \int_{t_0}^{t_f} L(\mathbf{x}, \mathbf{u}, t)dt \tag{3.13}$$

where ϕ describes the terminal cost function, and L is the accumulated cost function. The underlying principle is that in the absence of constraints, the Hamiltonian function $\mathcal{H}$ can

be expressed as

$$\mathcal{H} = L(\mathbf{x}, \mathbf{u}, t) + \boldsymbol{\lambda}^T \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{3.14}$$

where $\boldsymbol{\lambda}$ represents the vector of costates, which can change with respect to time. After some transformations, it follows that

$$\dot{x} = \frac{\partial \mathcal{H}}{\partial \boldsymbol{\lambda}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t) \tag{3.15}$$

and

$$\dot{\boldsymbol{\lambda}} = -\frac{\partial \mathcal{H}}{\partial \mathbf{x}} = -\frac{\partial L(\mathbf{x}, \mathbf{u}, t)}{\partial \mathbf{x}} - \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u}, t)}{\partial \mathbf{x}} \boldsymbol{\lambda} \tag{3.16}$$

The optimal control law is found by minimizing the Hamiltonian by adjusting the control, and as such, a control $\boldsymbol{u}$ must be found so that

$$\frac{\partial \mathcal{H}}{\partial \mathbf{u}} = 0 \tag{3.17}$$

and, to make sure the Hamiltonian is at a local minimum, the second derivative matrix must be positive definite:

$$\frac{\partial^2 \mathcal{H}}{\partial^2 \mathbf{u}} > [0] \tag{3.18}$$

In the Cartesian system, a spacecraft with a constant specific impulse can be described with the dynamics as described in section 3.2. Expanding those equations yields

$$\ddot{x} = -\frac{\mu x}{r^3} + \frac{T}{m} \cos \alpha_z cos\alpha_{xy} \tag{3.19a}$$

$$\ddot{y} = -\frac{\mu y}{r^3} + \frac{T}{m} \cos \alpha_z sin\alpha_{xy} \tag{3.19b}$$

$$\ddot{z} = -\frac{\mu z}{r^3} + \frac{T}{m} \sin \alpha_z \tag{3.19c}$$

where α_z is the elevation angle of the thrust vector as measured from the inertial x-y plane and α_{xy} is the azimuth angle of the thrust vector measured from the x direction. The

maximum rate of change of mass can be assumed to be a constant:

$$\dot{m} = -c = -\frac{T_{max}}{g_0 I_{sp}} \tag{3.20}$$

and is an essential part of the dynamical equations. These equations can be simplified so that the thrust and rate becomes nondimensionalized by taking the transformations

$$\frac{T}{m} = \tau \tag{3.21a}$$

$$\dot{\tau} = \frac{Tc}{m^2} = \frac{\tau^2 c}{T} = \tau^2 k \tag{3.21b}$$

where *k* is known as the thrust efficiency parameter. Due to these transformations, the second order dynamical system can be transformed into a nondimensional first-order system:

$$\dot{x} = v_x \tag{3.22a}$$

$$\dot{y} = v_y \tag{3.22b}$$

$$\dot{z} = v_z \tag{3.22c}$$

$$\dot{v_x} = -\frac{\mu x}{r^3} + \tau \cos\alpha_z cos\alpha_{xy} \tag{3.22d}$$

$$\dot{v_y} = -\frac{\mu y}{r^3} + \tau \cos\alpha_z sin\alpha_{xy} \tag{3.22e}$$

$$\dot{v_z} = -\frac{\mu z}{r^3} + \tau \sin\alpha_z \tag{3.22f}$$

$$\dot{\tau} = \tau^2 k \tag{3.22g}$$

From these equations, the Hamiltonian is seen to be:

$$\mathcal{H} = \lambda_1 v_x + \lambda_2 v_y + \lambda_3 v_z + \lambda_4(-\frac{\mu x}{r^3} + \tau \cos\alpha_z cos\alpha_{xy}) + \lambda_5(-\frac{\mu y}{r^3} + \tau \cos\alpha_z sin\alpha_{xy}) + \lambda_6(-\frac{\mu z}{r^3} + \tau \sin\alpha_z) + \lambda_7 \tau^2 k \tag{3.23}$$

The costate equations can be derived, knowing (3.16):

$$\dot{\lambda_1} = \lambda_4 \frac{y^2 + z^2 - 2x^2}{r^5} - \frac{3x(\lambda_5 y + \lambda_6 z)}{r^5} \tag{3.24a}$$

$$\dot{\lambda_2} = \lambda_5 \frac{x^2 + z^2 - 2y^2}{r^5} - \frac{3y(\lambda_4 x + \lambda_6 z)}{r^5} \tag{3.24b}$$

$$\dot{\lambda_3} = \lambda_6 \frac{y^2 + x^2 - 2z^2}{r^5} - \frac{3z(\lambda_4 x + \lambda_5 y)}{r^5} \tag{3.24c}$$

$$\dot{\lambda_4} = -\lambda_1 \tag{3.24d}$$

$$\dot{\lambda_5} = -\lambda_2 \tag{3.24e}$$

$$\dot{\lambda_6} = -\lambda_3 \tag{3.24f}$$

$$\dot{\lambda_7} = -\lambda_4 \cos(\alpha_z)\cos(\alpha_{xy}) - \lambda_5 \cos(\alpha_z)\sin(\alpha_{xy}) - \lambda_6 \sin\alpha_z - 2\lambda_7 \tau k \tag{3.24g}$$

As stated in (3.17) the derivative of the Hamiltonian with respect to control must equal zero. Therefore, it is known that:

$$\frac{\partial \mathcal{H}}{\partial \alpha_z} = 0 = -\lambda_4 \tau \sin(\alpha_z)\cos(\alpha_{xy}) - \lambda_5 \tau \sin(\alpha_z)\sin(\alpha_{xy}) + \lambda_6 \tau cos(\alpha_z) \tag{3.25}$$

Also, the second derivative of the Hamiltonian must be positive. The second derivative can be found to be:

$$\frac{\partial^2 \mathcal{H}}{\partial \alpha_z^2} = -\lambda_4 \tau \cos(\alpha_z)\cos(\alpha_{xy}) - \lambda_5 \tau \cos(\alpha_z)\sin(\alpha_{xy}) - \lambda_6 \tau sin(\alpha_z) > 0 \tag{3.26}$$

Due to this constraint, we know that:

$$sign(\lambda_6) = -sign(\sin(\alpha_z)) \tag{3.27}$$

$$sign(\lambda_4 \cos(\alpha_{xy}) + \lambda_5 \sin(\alpha_{xy})) = -sign(\cos(\alpha_z)) \tag{3.28}$$

Therefore, the optimal control for the thrust elevation angle is:

$$\alpha_z^* = atan2(\frac{-\lambda_6}{-(\lambda_4 \cos(\alpha_{xy}) + \lambda_5 \sin(\alpha_{xy}))}) \tag{3.29}$$

Similarly, for the azimuthal control, the derivative of the Hamiltonian is:

$$\frac{\partial \mathcal{H}}{\partial \alpha_{xy}} = 0 = -\lambda_4 \tau \cos(\alpha_z) \sin(\alpha_{xy}) + \lambda_5 \tau \cos(\alpha_z) \cos(\alpha_{xy}) \tag{3.30}$$

The second order derivative is:

$$\frac{\partial^2 \mathcal{H}}{\partial \alpha_{xy}^2} = -\lambda_4 \tau \cos(\alpha_z) \cos(\alpha_{xy}) - \lambda_5 \tau \cos(\alpha_z) \sin(\alpha_{xy}) > 0 \tag{3.31}$$

Subsequently, the two constraints for the azimuthal control angle that come from the second order derivative

$$\alpha_{xy}^* = atan2(\frac{-\lambda_5}{-\lambda_4}) \tag{3.32}$$

However, this is not a guarantee that the trajectory found is truly mass-optimal due to the fact that the costates are guessed. Additionally, the control generated is quadratic and is therefore suboptimal. Homotopy, also known as embedding, is used to transform the control to become mass-optimal. In general, homotopy seeks to solve a function:

$$\mathbf{f}(\mathbf{x}) = 0 \tag{3.33}$$

by starting with a known solution to a set of problems:

$$\mathbf{h}(\mathbf{x}, \alpha) = 0 \tag{3.34}$$

which, when $\alpha = 0$, is a problem that is relatively easy to solve, with a solution $\boldsymbol{x_0}$, and progressively increasing α until

$$\mathbf{h}(\mathbf{x}, 1) = 0 \tag{3.35}$$

can be found, since the resulting equation becomes 3.33. With every increase in α the last solution $\mathbf{x}$ is used as an initial guess. 0nce the control becomes mass-optimal, the bang-bang control structure may be seen, where the throttle rapidly switches from maximum thrust to no thrust and vice versa. An example of the control for a transfer used in the training data can be seen in Figure 4.1.

3.6 Design of Experiments

In order to sufficiently analyze the large design space of possible flyby orbit transfers, a design of experiments methodology was used. This is because the definition of two orbits are dependent on a total of 12 orbital elements, and when considering the many combinations of orbits and combinations of orbital elements, it is apparent that the design space is truly expansive. In addition, the large computational difficulty of finding feasible transfers limits the number of orbital transfer runs that can be run. Therefore, the design of experiments methodology was used, as it minimizes the number of computational runs while maximizing the information that can be gleaned from the results[41]. There are many types of design of experiments, such as Full Factorial, Latin Hypercube, Box-Behnken, and Central Composite. They each have their advantages and disadvantages, so the choice of design of experiments must be made while considering the problem design space at hand. A Latin Hypercube design of experiments, seen in Figure 3.5, was chosen to explore the space of potential orbital transfers. The Latin Hypercube arrangement was chosen due to its good coverage of the interior of the design space, as it provides a balance of minimal runs and uniformity[41]. The main disadvantage of the Latin Hypercube, poor coverage at the corners of the design space, can largely be ignored for the problem investigated in this

thesis since it can be negated by choosing slightly larger bounds on the design variables.

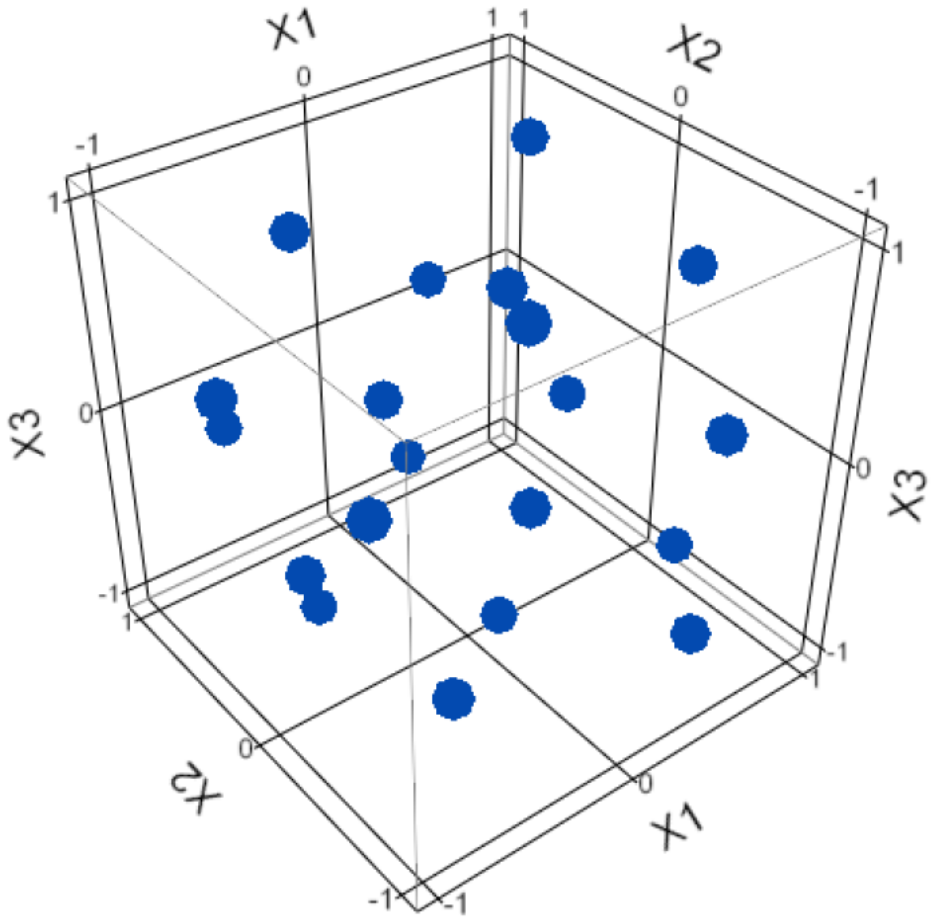


Figure 3.5: The Latin Hypercube Design of Experiments (from [41])

CHAPTER 4
EXPERIMENTS

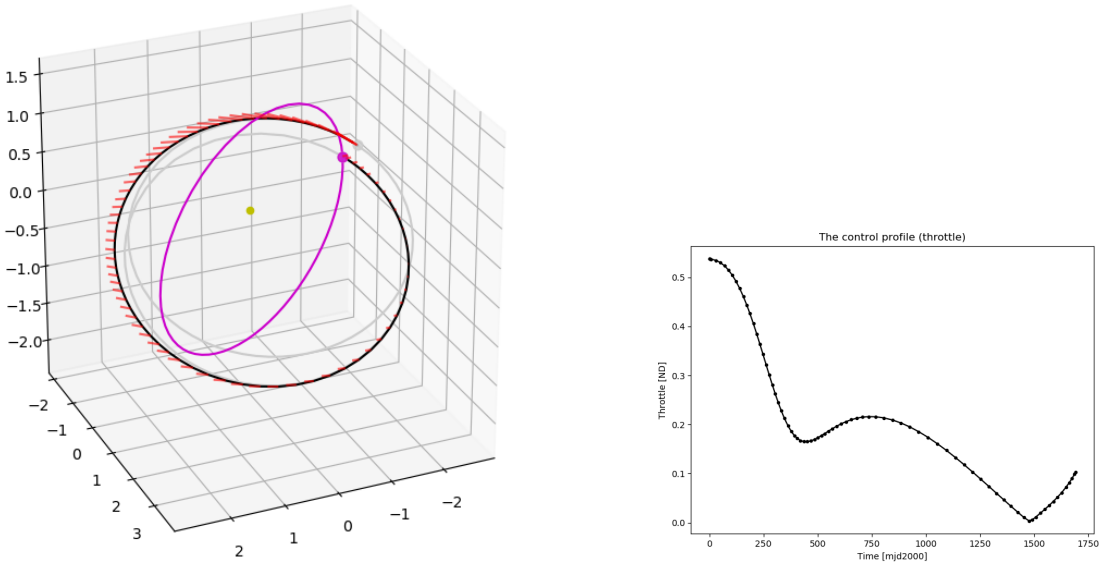
4.1 Computational Setup

To give context to computation times presented in this thesis, the hardware and software characteristics of the computational setup are described here. All experiment runs were conducted on a Windows 10 laptop computer with a Intel(R) Core(TM) i7-7500U CPU at 3 GHz, with 2 cores and 4 threads. It was observed that during all runs, the program was bottlednecked by CPU speed, as CPU utilization was typically around 75%, while RAM usage stayed much lower. All computation times provided in this thesis, unless otherwise noted, are with all cores utilized, through the use of Python multiprocessing, which allows for parallel computation.

4.2 Experiment for Research Question 1

As mentioned in the previous chapter, a determination was made with the results of the experiment to use a direct method for creating the training data. The reasons for the determination are also in the previous chapter, and are described here in more detail. Using Pontryagin's principle and homotopy, a portion of the feasible trajectories found via the direct method were used as initial guesses for the indirect method to see if this would be a computationally-efficient way of finding near-optimal trajectories, thus populating the training data. For those that were nearly time-optimal to begin with, the effect of homotopy to move from quadratic control to bang-bang control was not noticeable (see Figure 4.3) since the original trajectory was already thrusting for nearly the entire route. Additionally, there was a lack of freedom in designing the indirect trajectories, since for the implementation in PyKEP, all spacecraft states need to be defined for both the departure and the arrival.

This meant that, more often than not, using the indirect method could not improve on the optimality of the solution. In some cases, the final predicted mass for the indirect solver was lower than that predicted by the direct solver. The indirect method solutions are treated in this thesis as a higher-fidelity version of the results, and not necessarily more optimal, since the Sims-Flanagan results being more optimal may be unrealistic - resulting from the simplifying assumption of small, impulsive maneuvers of the Sims-Flanagan method. Meanwhile, the indirect method models the continuous thrust on the entire trajectory. Also, since the costates in this implementation were initialized with random numbers, and in accordance with the fact that indirect methods are sensitive to the initial guesses, the runs were found to be highly stochastic in nature. It was observed that if the same initial input was repeated, some runs would result in infeasible trajectories and others in feasible ones. Due to the long computation time (average of 3 minutes per trajectory) to find the indirect trajectory, each feasible trajectory was only run through the indirect solver once, whether or not the indirect solver found a feasible trajectory. Unfortunately, this significantly cut down the number of indirect trajectories that are incorporated in this thesis. Only 36 out of 100 feasible direct trajectories input converged to become feasible indirect trajectories. An example run of the trajectory produced by the indirect solver can be seen in Figure 4.1. As surmised earlier, the lack of mass optimality seen in the lack of bang-bang topology in the control for the transfer may be attributed to the lack of design freedom of the indirect solver. This is akin to an optimizer being given an initial guess close to a local optimum (analogous to the solution given by the direct solver). Secondly, the indirect optimizer was liable to show that the previously found direct trajectories to be unfeasible since the simplifying assumptions involved in the Sims-Flanagan solver may slightly overestimate the performance of the spacecraft. This is because of the small impulsive burn assumption as stated previously. Since the indirect solutions showed little difference when compared to the direct solutions when comparing final mass, in addition to the fact that time-of-flight was fixed between the two types of methods, it was decided that using a direct method



(a) A continuous-thrust transfer. Red lines indicate thrust direction and magnitude.

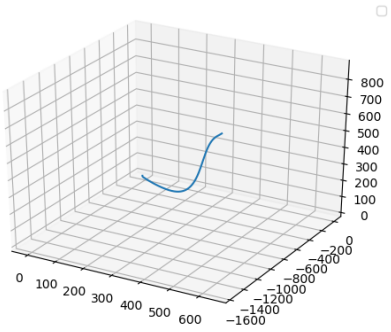
(b) The throttle control for the transfer.

Figure 4.1: An example run of the indirect solver.

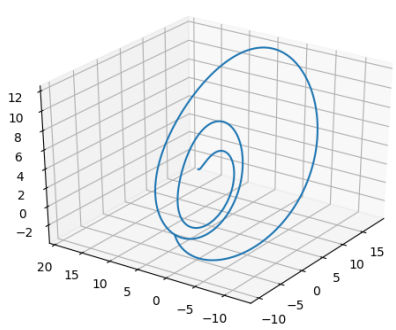
alone to generate training data was a more pragmatic choice, because of the much larger computation time associated with the indirect solver. Therefore, **Research Question 1a** was formulated to decide which direct method should be used to create the training data.

4.3 Experiment for Research Question 1a: Choice of Direct Solver

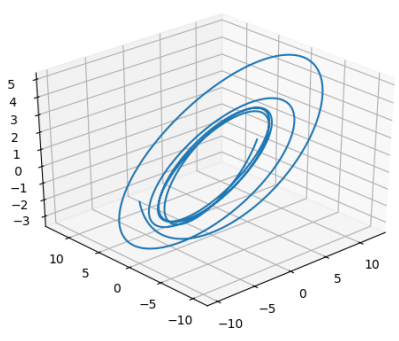
Initial experiments indicated that the Chebyshev polynomial method was slower compared to the more tried-and-true low-thrust trajectory solvers found in the Python package PyKEP [37]. It was found that finding a 11^{th} order Chebyshev trajectory would take around 4 minutes of computation, due to the looped optimization of the Chebyshev coefficients that need to be found via sequential quadratic programming (SQP). The computation time does not include the additional effort to transform the Chebyshev polynomial trajectory to feasible trajectory via the use of the generalized Newton method. The computational difficulty of this method was further exacerbated by the sensitivity of the quality of the created trajectory to the maximum order of the Chebyshev polynomial used in modeling the trajectory. In order to illustrate this, Figure 4.2 show the difference in trajectory in a notional trans-



(a) Trajectory using 4^{th} order Chebyshev polynomial.



(b) Trajectory using 10^{th} order Chebyshev polynomial.



(c) Trajectory using 11^{th} order Chebyshev polynomial.

Figure 4.2: Chebyshev polynomial-based trajectory depends heavily on the order of the polynomial.

fer when limiting the Chebyshev polynomial order to 4, 10, and 11. It can be seen from the trajectories generated that the 10^{th} order trajectory traveled fewer than 3 revolutions before arriving, while the 11^{th} order trajectory orbited for over 6 revolutions. This sensitivity would increase the computational difficulty of finding a feasible trajectory since more candidate trajectories need to be attempted, since it would be hard to draw a line such that no extraneous runs of higher-order orbit, with its longer computation time, would be conducted if a lower-order run could be made to converge to a realistic orbit. The Chebyshev polynomial method as implemented by the author was found to be slower than PyKEP's implementation of the Sims-Flanagan low-thrust optimizer, whose runs typically

lasted around 30 seconds each, compared to 2-5 minutes for the Chebyshev polynomial method, depending on the maximum order of the polynomials. The comparison of computation time between the author's implementation of Patel's Chebyshev polynomial method and PyKEP's Sims-Flanagan is not altogether fair due to the fact that the author coded the Chebyshev method in Python while PyKEP[42] is mainly written in C++ and is optimized for speed. This is supported by the fact that Patel's code, when written in C++ and Objective C, was able to compute a Earth-Mars trajectory (albeit with unoptimized time-of-flight and launch date) in an average of 0.1 seconds[35]. Meanwhile, the same transfer would take around 30 seconds in PyKEP, where it would be an optimized trajectory in a bounded range time-of-flight and launch date. For the problem described in this thesis, PyKEP's Sims-Flanagan implementation was deemed a better choice for creating the training data for the machine learning algorithm, since it would be faster to use PyKEP rather than to use the Chebyshev polynomial method to populate a search space and then to find the global optimum within that search space. PyKEP's Sims-Flanagan solver was helped by the inclusion of SNOPT7, or Sparse Nonlinear OPTimizer, which is an implementation of SQP particularly well-suited for optimizing large, computationally-expensive nonlinear problems[43][44]. SNOPT7 was used since trial runs comparing it with another implementation of SQP, namely SLSQP, or Sequential Least SQuares Programming, from the Python package SciPy, indicated that the former, although slightly slower per trajectory analyzed, would produce a greater proportion of feasible results than running SLSQP. Trial run results can be seen in Table 4.1. A small decrease in performance of around 25% was deemed acceptable due to the sizable increase in training data the use of SNOPT7, as opposed to SLSQP, would provide. The difference in computation speed is likely due to a higher iteration limit used for the SNOPT7 solver, at 2,000 major iterations and 200,000 minor iterations, compared to the 50,000 iteration limit used by SLSQP in this work, rather than any comparative inefficiencies in SNOPT7.

Table 4.1: Comparison between trial runs of 100 object pairs solved using SLSQP vs SNOPT7

	SLSQP	SNOPT7
Minutes to run	~24	~30
Number of feasible trajectories found	58	83

4.4 Experimental Setup

When running SNOPT7, the feasibility tolerance parameter, which controls the tolerance for mismatches in Sims-Flanagan match points, was set to a value of 1e-4. This allows a difference of 1e-4 m and 1e-4 m/s for the mismatches in position and velocity, respectively, at each match point. One of the more important choices during the experiment setup is the length of the launch (or departure) window, which is the time period during which the spacecraft can begin thrusting on its way to a target. Due to limitations in computational power, the departure date window was limited to a range of 6,000 days, as a longer window would cause the optimization algorithm to try more trajectories possibilities between two objects. In previous work in trajectory analysis, such as the often-investigated Earth-Mars transfer[45], the analysis window was set to the synodic period of the two orbits. A synodic period is the period in time in which two objects orbiting a central body have a conjunction, or line up. Analyses using the synodic period rely on the assumption that the two orbits are coplanar and circular[46]. For most low-fidelity trajectory analyses between planets, these assumptions are not unreasonable, and a trajectory optimizer need only look at a departure window of one synodic period. However, the departure window chosen in this thesis is longer than the synodic period of the two orbits precisely because these assumptions are not valid for the problem described in this thesis, where transfers between highly noncoplanar and elliptical orbits are possible. The period where two objects are guaranteed to return to the same place relative to each other is the least common multiple of the periods of the

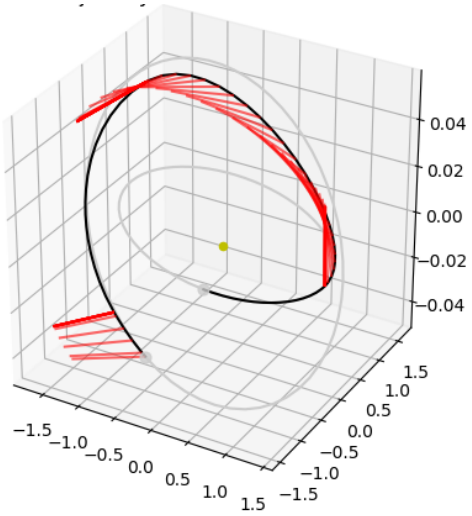
Table 4.2: Comparison between trial runs of 100 object pairs solved while varying basin hopping settings

# of consecutive basin hopping runs allowed without improvement	# of feasible trajectories found	Computation time (minutes)
1	43	10
2	63	22
5	77	45

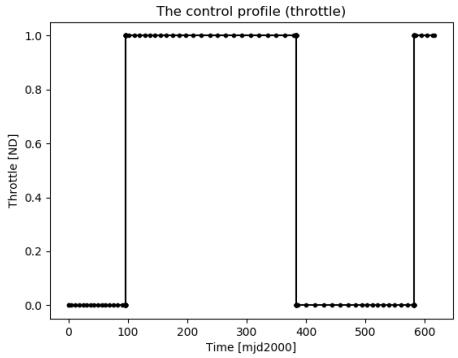
two objects, which can be extremely large. For lower-fidelity analyses, mission designers may round the orbital periods so that the least common multiple of the periods is lower than simply the multiple of the periods. For example, Earth-Mars cycler analyses assume that Mars' orbit is exactly $1\frac{7}{8}$ Earth years, allowing for the assumption that the orbits inertially repeat every 15 years, while the actual orbit period is around 1.881 years[46]. For the level of analysis in this thesis, and taking into account that the maximum orbital period involved in this thesis is less than 1,900 days, a departure window of 6,000 days was considered an acceptably lengthy interval. Another parameter that had a large influence on the computation time was the number of consecutive basin hopping iterations allowed without showing improvement. This value also had a large influence on the number of feasible trajectories produced. Trial runs of 100 flyby transfers were undertook to ascertain the sensitivity of the computation time and feasible trajectory percentage to the basin hopping setting. The results of these trial runs can be seen in Table 4.2. Subsequently, the number of consecutive iteration was limited to 3 as it appeared to be an acceptable trade between number of trajectories found and speed of computation.

4.5 Indirect Solver

For the indirect solver, another program from PyKEP was used. The main purpose of running direct solver trajectory solutions into the indirect solver was to put some context to the amount of errors in the machine learning algorithm. The indirect solver in PyKEP The costates were randomly generated and the initial and final states are fixed. The indirect



(a) A continuous-thrust transfer. Red lines indicate thrust direction and magnitude.



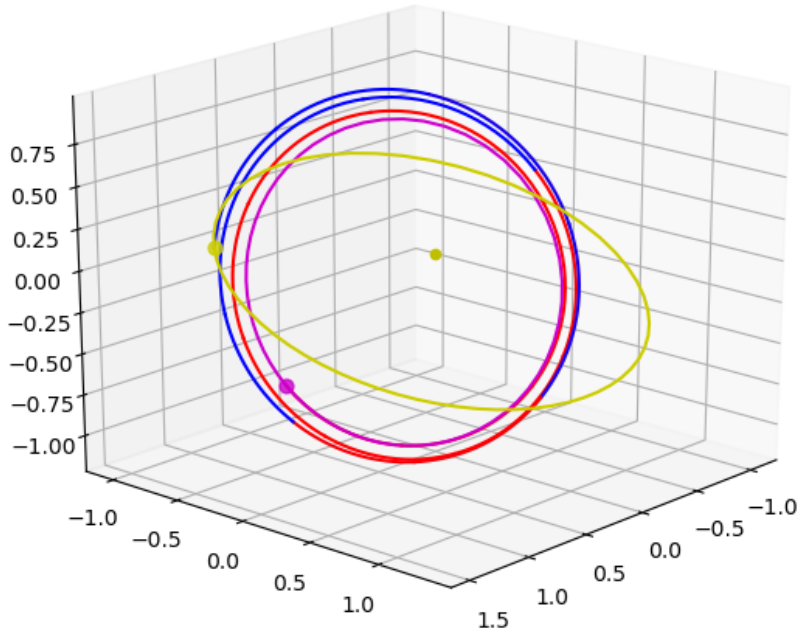
(b) The throttle control for the transfer.

Figure 4.3: Indirect solver's solution to the Earth-Mars rendezvous problem.

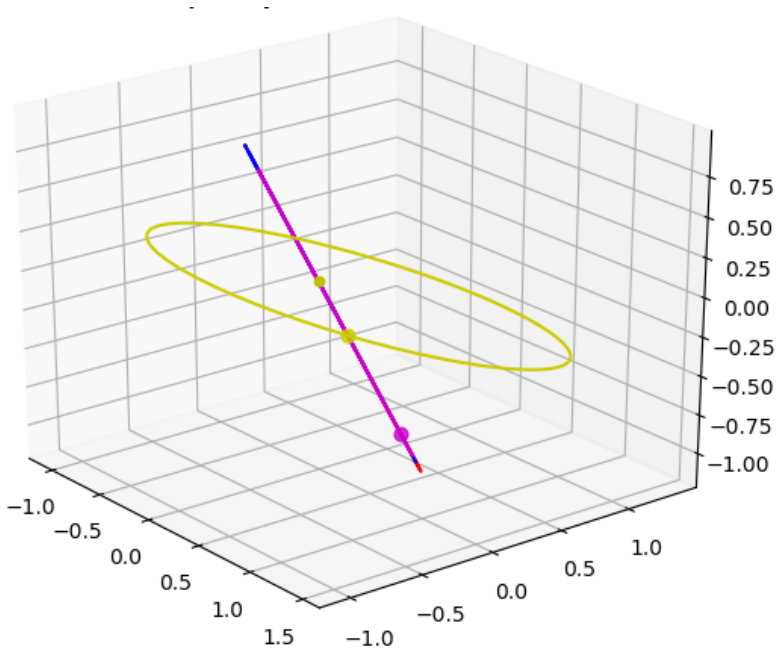
method then solves for the mass-optimal trajectory under these constraints.

4.6 Training Data

Upon inspection, orbit solutions produced by the PyKEP implementation of Sims-Flanagan are realistic. Significant portions of the orbit are in thrusting, and the orbit is changing correctly. Moreover, the trajectories are fairly time-optimal as the spacecraft is thrusting for most of the orbit, as evidenced in [37] and [47]. As allowed by the scenario setup, the spacecraft is intercepting the target at a high relative velocity, and leaving the original object with no relative velocity. Furthermore, the satellite does not diverge significantly from the plane of the original object, which is advantageous since even a small divergence would result in a large fuel consumption. A multiple-revolution orbit transfer can be seen in Figure 4.4. When all transfers used for training are considered, the time-of-flight and fuel expenditures are distributed as seen in Figure 4.5. Compare with the figure found in [37], which describes transfers between Earth and NEAs. Since mass-optimality was used for the creation of the training data, there is no clear line of time-optimal transfers (where the thrust is on for the entire transfer). Also, it can be seen that for the training data set, the



(a) A typical flyby transfer. The spacecraft travels nearly three revolutions before flying by the target.



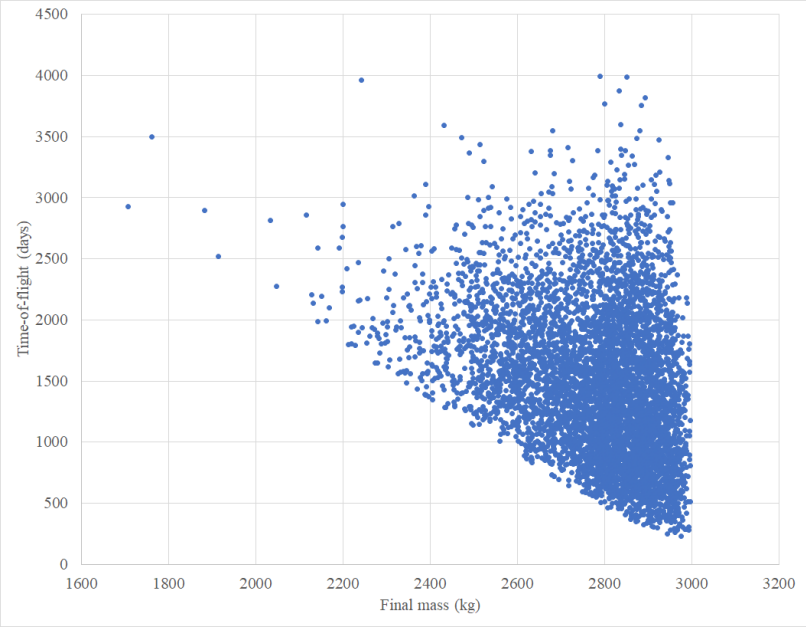
(b) The same transfer, with the original orbit viewed edge-on

Figure 4.4: A transfer between objects with a large difference in inclination. The spacecraft starts at the magenta point and travels to the yellow point. Red arcs indicate thrusting segments and blue arcs indicate coasting segments. Axes are in units of AU and to scale with each other.

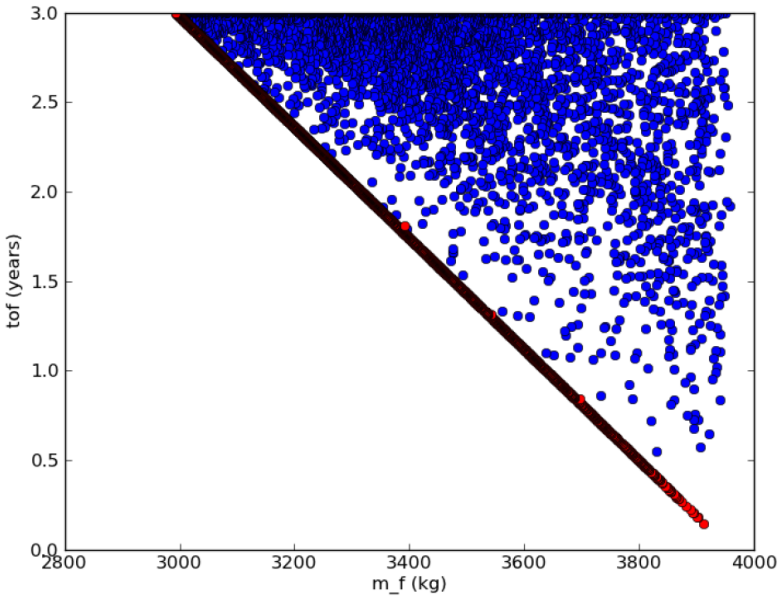
mass optimal trajectories tend to be skewed for higher final mass and lower time of flight. This appears to be a characteristic of flyby trajectories in particular, as it matches up with the observations in [47].

4.7 Experiment for Research Question 2

In order to see which machine learning algorithm produces the smallest errors, all six machine learning algorithms mentioned in this thesis were implemented and tested using the same set of training and validation data. Even though the design space is much larger than the one in [10], this work used a comparably smaller amount of training and validation data due to the large computational burden of creating trajectories even when using the Sims-Flanagan method. All six machine learning algorithms were run using the implementation from the Python package *scikit-learn*[28]. A total of 7,218 Sims-Flanagan trajectories were found - 5,774, or about 80% were used as training data for the machine learning algorithms. The same randomization of training and validation data was used between the different algorithms. The predictors for the machine learning algorithms were the orbital elements for the departure and arrival objects, 12 inputs in total. The outputs consisted of the final mass of the spacecraft and the time-of-flight of the transfer. In the meantime, all 7,218 trajectories were used to find the MAE and the root-mean-square error (RMSE) for the Lambert predictor. Different settings were used for the two ensemble methods involved in this thesis, random forest and gradient boosting. For the random forest method, the number of estimators was set to 100, while 35 estimators were used for the gradient boosting. In gradient boosting, a learning rate of 0.1 and a maximum depth of 3 were used. The artificial neural network consisted of an input and output layer of 12 and one neuron(s) respectively. Three hidden layers were used, numbering in 80, 80, and 50 in order from input to output. The computational effort needed to train the 6 machine learning methods are all small compared to that required to create any one trajectory. Within the machine learning algorithms, the artificial neural network took the longest to train, at around 3 sec-



(a) Graph for all 5774 feasible transfers in the training data.



(b) Graph for all found transfers from Earth to NEAs (from [37])

Figure 4.5: Time-of-flight vs. final spacecraft mass graphs

onds. Minimal optimization or tuning of the machine learning occurred, but the number of hidden layers in the ANN was set to 3 to prevent overfitting. Histograms displaying the mass error between machine learning estimates and Sims-Flanagan results for validation trajectories can be seen in Figure 4.6, Figure 4.7, and Figure 4.8. Table 4.3 contains the MAE and RMSE for the mass errors of machine learning methods and the Lambert prediction. It can be seen that any of the machine learning methods performed much better than the Lambert predictor. However, there is still significant error in the mass estimations for the machine learning methods. Therefore, it can be seen that the machine learning method is not meant for direct use in spacecraft design, but is more fitting for narrowing down destination choices. For time of flight, the validation data error histograms can be seen in Figure 4.9, Figure 4.10, and Figure 4.11. Comparison to the Lambert predictor could not be made since one of the inputs to the Lambert predictor is the time-of-flight, since a Lambert predictor's purpose is to take two points on an orbit and a time-of-flight to create a valid connecting orbit. However, it can still be seen that the machine learning regressions for time-of-flight are noticeably worse than that for the final mass estimation. The range of time-of-flight values in the training data was over twice the range of that of final mass, so it was expected that the errors of the machine learning algorithms for predicting time-of-flight, seen in Table 4.4 were higher. Notwithstanding, the errors for time-of-flight were still disproportionately high: the best-performing mass predictor had an average of 3.12% error, while time-of-flight had 29.5% error. This may indicate the need for additional predictors for more accurate time-of-flight prediction. Further research in enhancing the prediction accuracy is beyond the scope of this thesis as the ability to predict time-of-flights within the scale of the actual values of time-of-flights has been demonstrated. In other words, the MAEs of the time-of-flights for all neural networks evaluated were found to be lower than the average time-of-flight in the found trajectories.

In order to contextualize the errors produced by the machine learning prediction, trajectory solutions of higher fidelity than the solver used for training data were found. First,

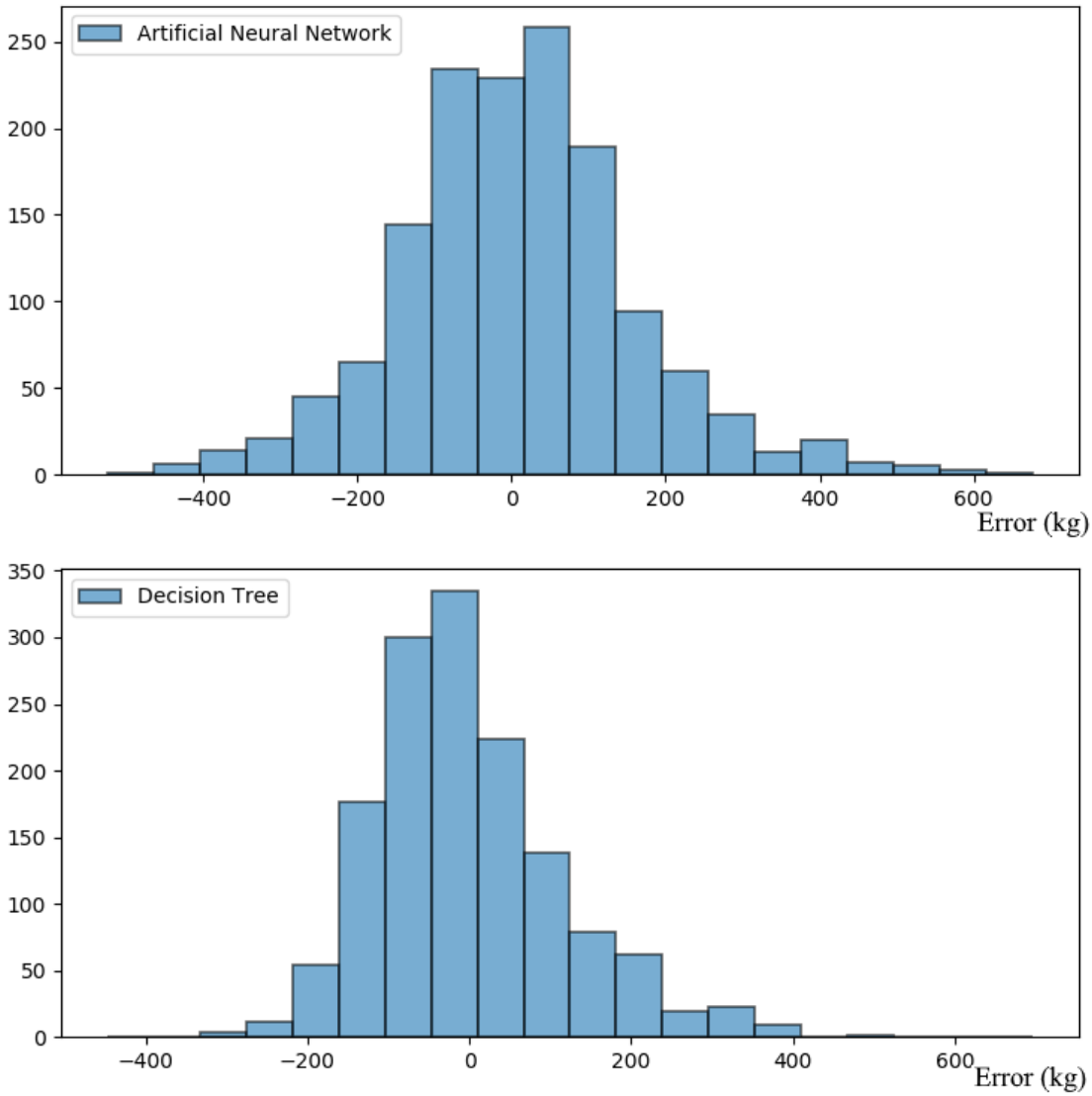


Figure 4.6: Validation trajectory mass errors for artificial neural network and decision tree algorithms.

trajectories found using the indirect method are compared to those in the training data. Also, the results of the Sims-Flanagan solver using the impulsive thrust assumption (the assumption used in the training data) were compared to the results found by running the Sims-Flanagan solver while considering continuous thrust.

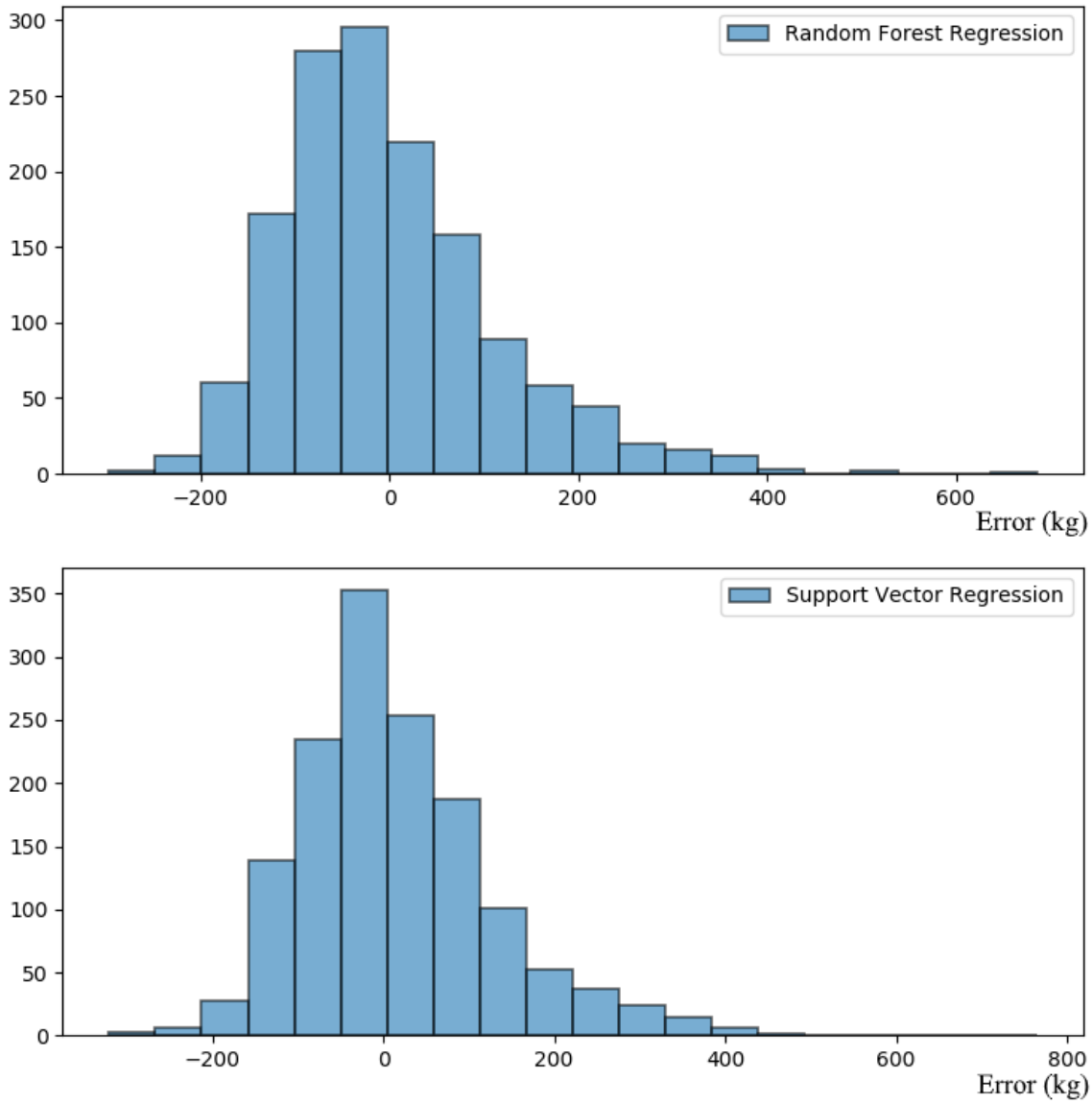


Figure 4.7: Validation trajectory mass errors for random forest regression and support vector regression.

Indirect Solver

After the indirect solver was run for the direct trajectory solutions, it was found that only about 36% of direct solutions resulted in a valid indirect solution. Comparing the differences in final vehicle mass between the valid indirect solutions and the corresponding direct solutions, it was found that the MAE for the direct solutions was 44.078 kg and the RMSE was 5442.891 kg. This is likely an underestimate of the true difference between the direct

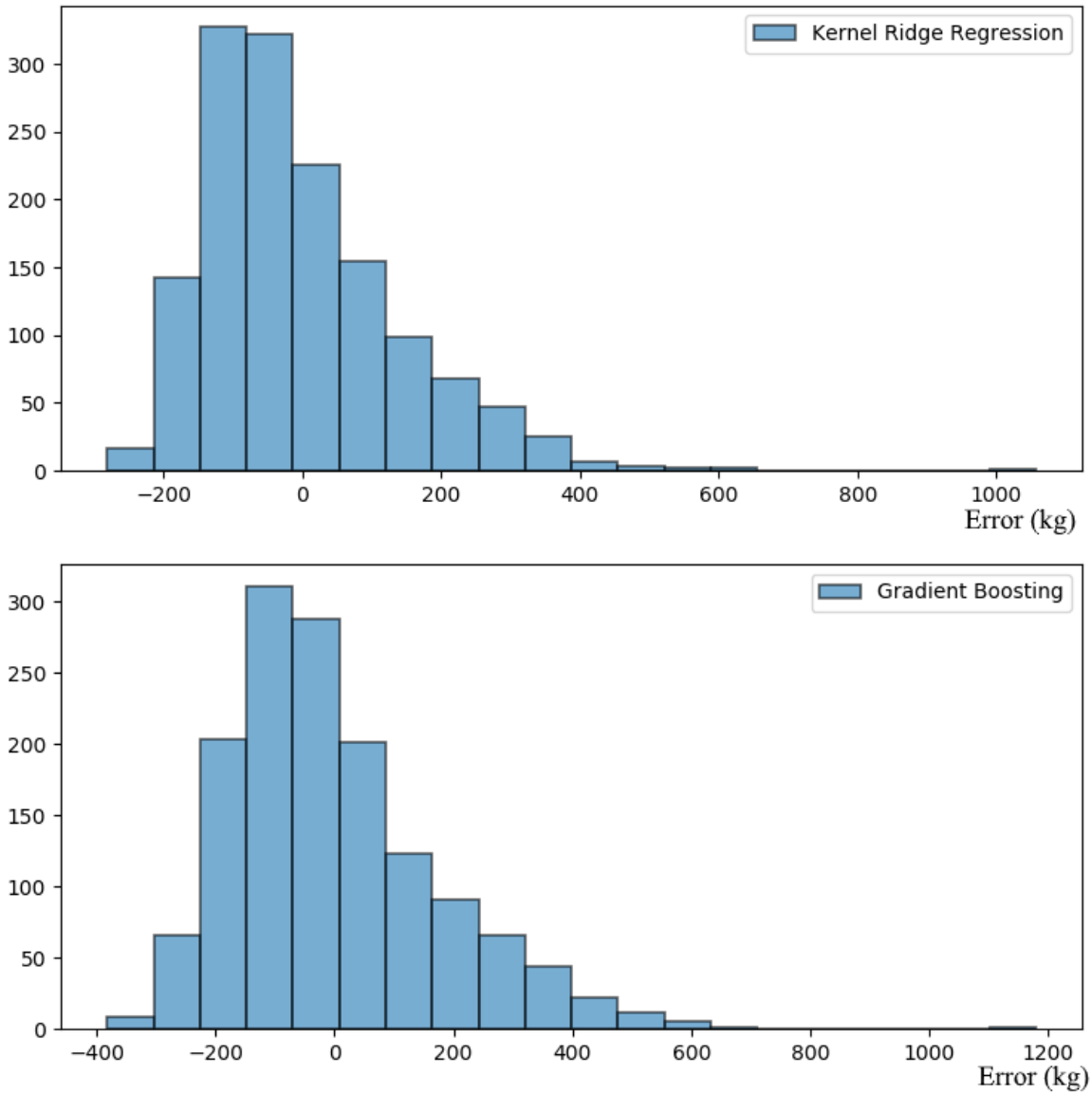


Figure 4.8: Validation trajectory mass errors for kernel ridge regression and gradient boosting.

solver and the indirect solver because unfeasible indirect results stemming from feasible direct results may be indicative of a potentially large error between the indirect and direct trajectories. What was unexpected was that nearly all of the indirect trajectories had a final mass lower than that of the direct trajectories, meaning that the direct trajectories were in fact underestimating the fuel required to reach destinations by a small amount. This was likely because of the simplifying assumptions made in the Sims-Flanagan approach. Seeing

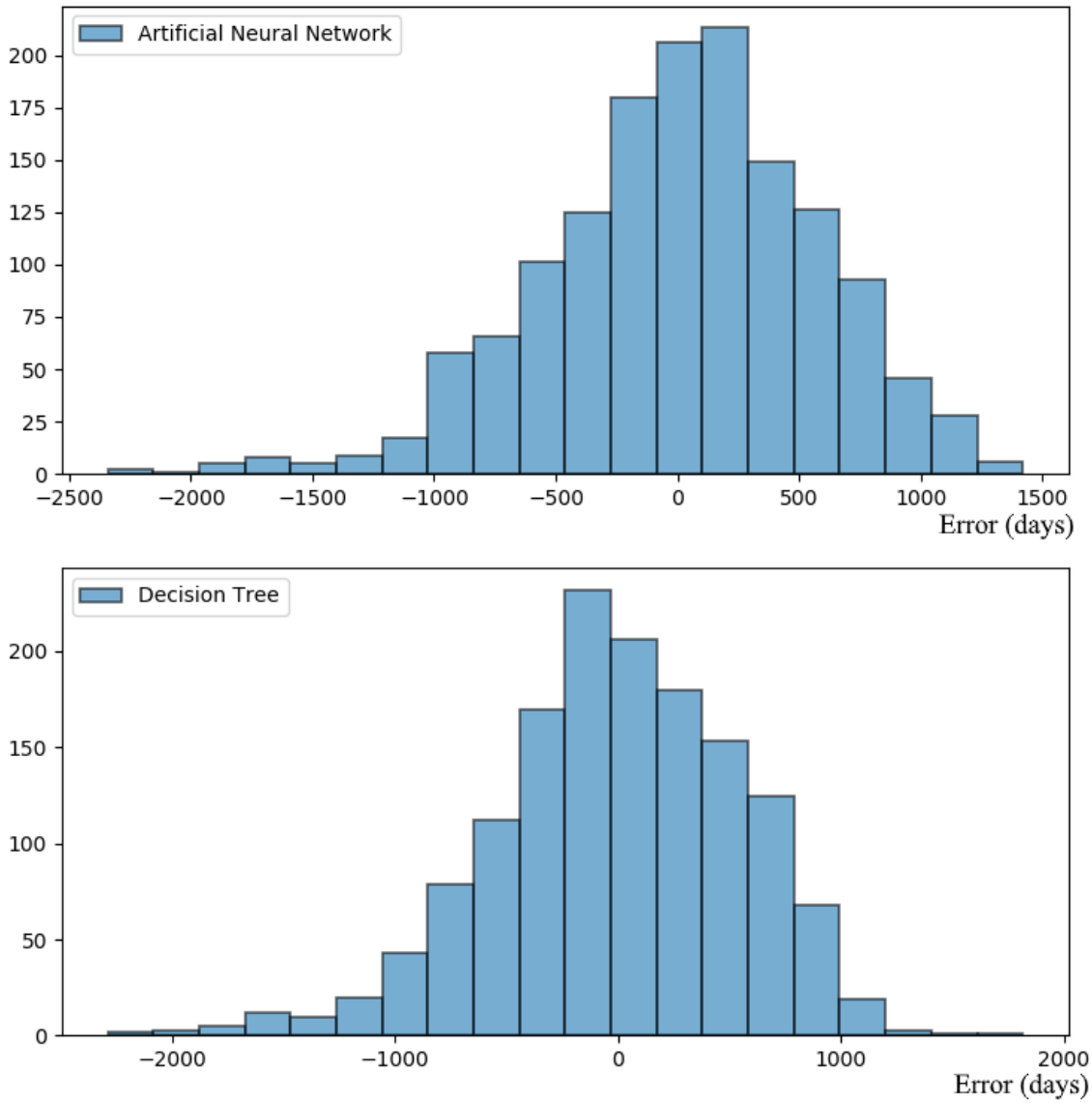


Figure 4.9: Validation trajectory mass errors for artificial neural network and decision tree algorithms.

that all the machine learning MAEs were within 3 times that of the direct solver itself, with the exception of gradient boosting, gives credence to the author's assertion that machine learning methods would be useful in low-fidelity use cases such as design space pruning for GTOC-class problems.

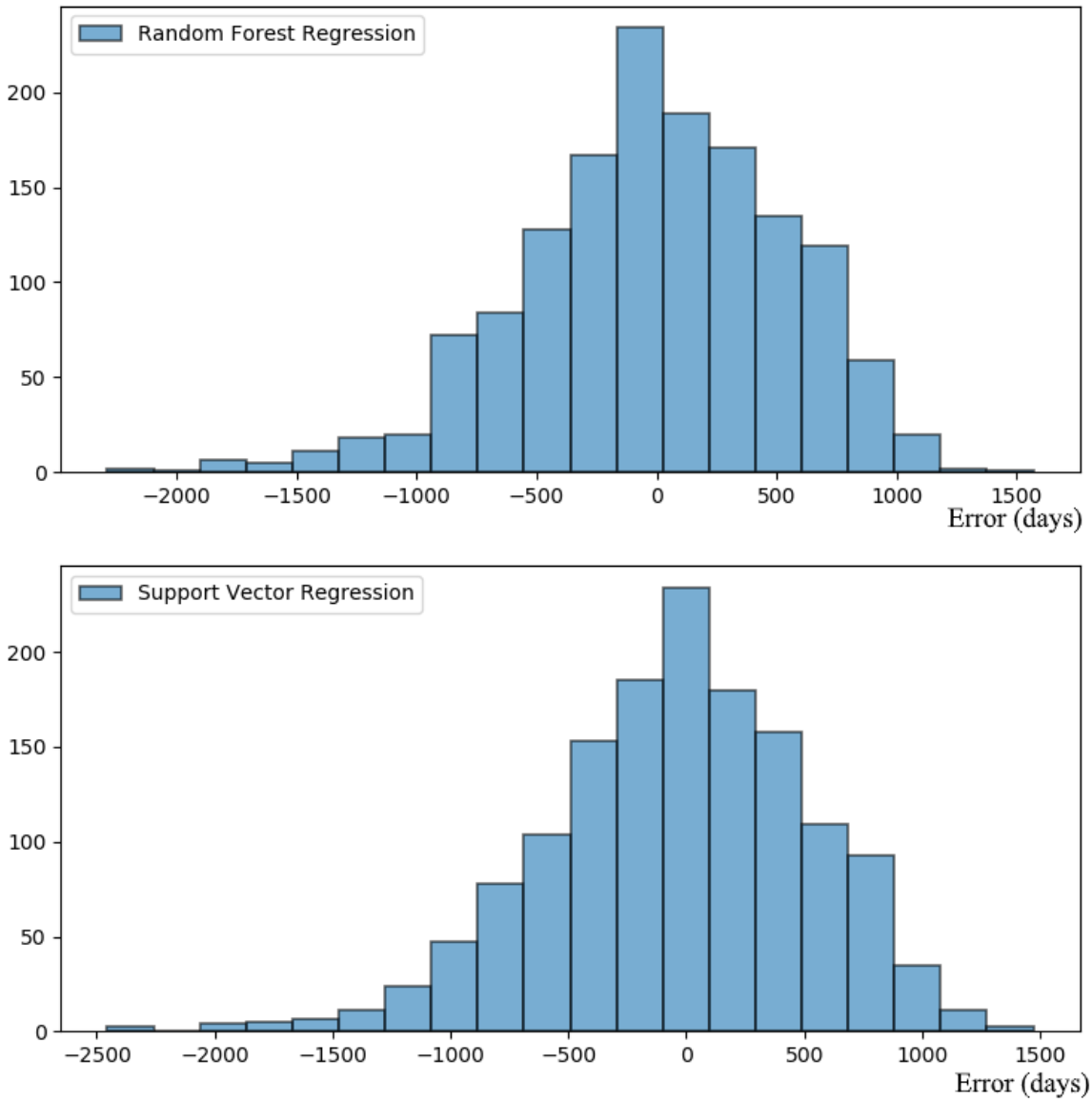


Figure 4.10: Validation trajectory mass errors for random forest regression and support vector regression.

Sims-Flanagan Method (Modeling Continuous Thrust)

PyKEP's implementation of the Sims-Flanagan method offers a choice between using the computationally-cheaper, lower fidelity method that assumes small impulsive burns, and a more computationally-intensive, higher fidelity method that simulates continuous thrust on the trajectory segments. The difference in final mass and time-of-flights for midpoint impulsive vs. continuous thrust were quantified through experiment. It was found that out

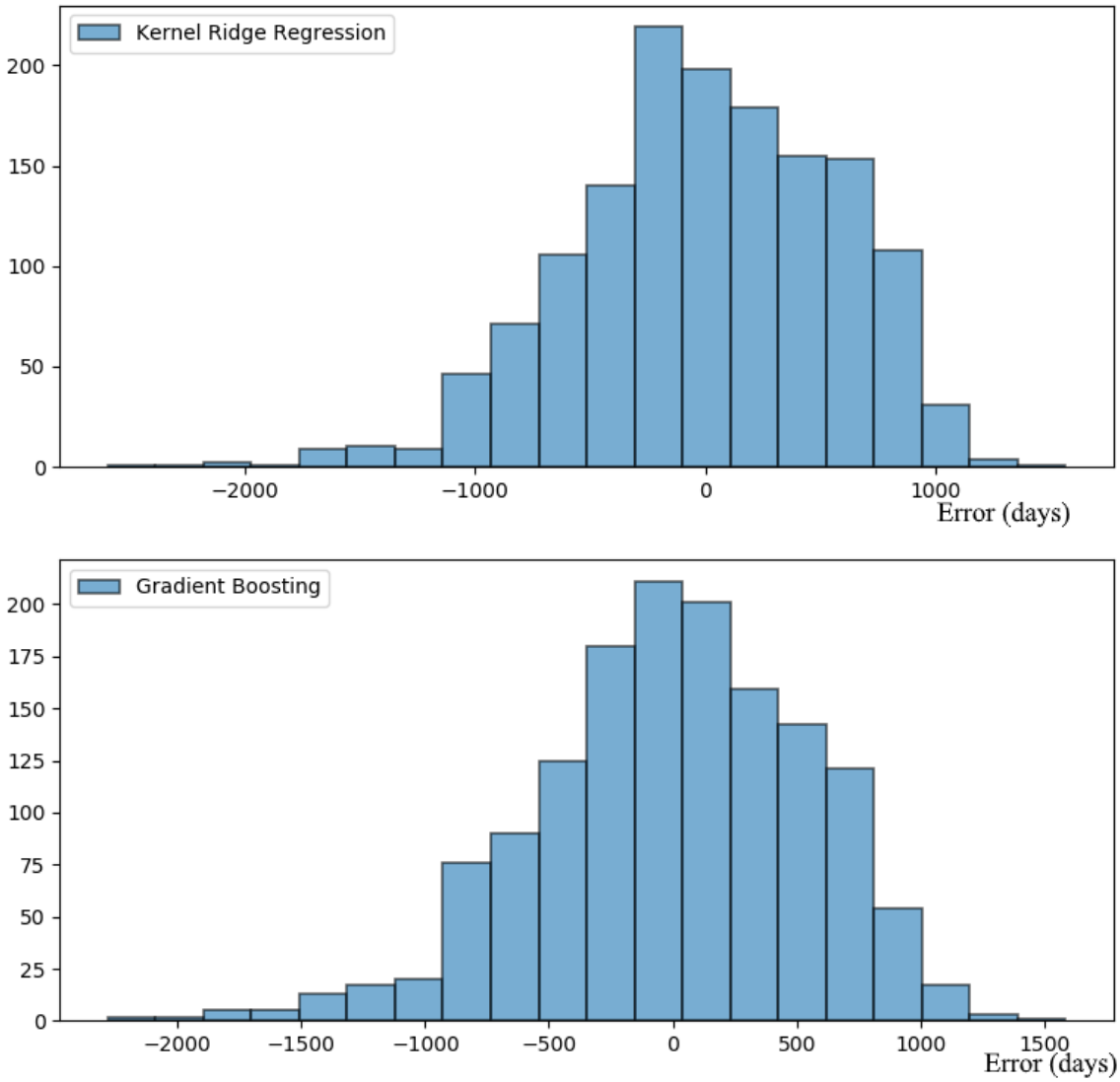


Figure 4.11: Validation trajectory mass errors for kernel ridge regression and gradient boosting.

of the total of 10,000 potential trajectories solved, 6,013 were feasible for both high and low fidelity runs. The mean absolute error (MAE) for the feasible trajectories was 73.73 kg for the mass and 462.35 days for the time-of-flight. These differences were partly because of the stochastic manner in which the overall trajectory optimization is conducted, particularly in the monotonic basin hopping. Narrowing down the matching feasible trajectories to those where the time-of-flight difference was within 100 days produced a subset of trajectories numbering 1,739. In this particular subset, the MAE for final mass was much lower,

Table 4.3: MAE and RMSE of the machine learning algorithms compared to that of the Lambert predictor for final spacecraft mass

	MAE (kg)	RMSE (kg)
Artificial Neural Network	116.515	23879.675
Decision Tree	90.879	14348.804
Random Forest Regression	98.984	16622.83
Support Vector Regression	86.525	13994.581
Kernel Ridge Regression	110.48	20031.551
Gradient Boosting	130.237	28073.997
Lambert Predictor	921.075	984759.561

Table 4.4: MAE and RMSE of the machine learning algorithms for time-of-flight

	MAE (days)	RMSE (days)
Artificial Neural Network	629.296	645475.1
Decision Tree	436.723	307267.304
Random Forest Regression	450.424	317205.096
Support Vector Regression	441.448	319474.413
Kernel Ridge Regression	455.4	321268.11
Gradient Boosting	432.092	297753.254

at 30.49 kg, and the MAE for time-of-flight was much lower than the imposed limit of 100 days, at 27.27 days. The distribution of absolute time-of-flight errors was skewed towards the lower end, implying that the monotonic basin hopping algorithm "pushed" solutions towards the same basins, as well as indicating that the difference in results between high and low fidelity modes of the Sims-Flanagan method were not great when compared to the effect of monotonic basin hopping. These results support the hypothesis that the machine learning results are reasonably accurate since the overall MAE between the continuous and impulsive-thrust Sims-Flanagan trajectories is in the neighborhood of the MAE for the machine learning methods, especially when the effect of monotonic basin hopping is considered. Lastly, the effectiveness of machine learning methods in this thesis cannot be directly compared to the effectiveness of those methods in previous literature, which dealt with transfers between Earth and NEAs[10]. Although both implementations of machine learning, in [10] and this thesis, solve for final mass, the former defined the scenario such that the spacecraft's dry mass is known, whereas in the latter, the initial mass is known.

Also, the machine learning algorithms in previous literature were trained on a set of 50,000 rendezvous transfers, compared to the 7,218 flyby transfers in this thesis. However, a rough comparison can be made by looking at the ratio of MAE in the mass prediction using the Lambert predictor to the MAE of the best-performing machine learning method. In [10], that ratio was around 71.96. In this thesis, the ratio was approximately 10.65. The accuracy gain from using a machine learning method is not as large in this thesis, but it is still in the range of that in previous work. Due to the large differences in scenario and training data, the comparatively lower ratio in MAE does not by itself refute the hypothesis for **Research Question 2**. The observed errors of the machine learning methods were on the scale of errors between direct and indirect method solutions, and errors between impulsive and continuous-thrust implementations of the Sims-Flanagan method. Therefore, it is reasonable to say that the machine learning methods can predict a transfer's time and fuel costs, for a larger design space, while experiencing error not much larger than the difference between a low and mid-fidelity solver. A summary of the methodology undertook in this thesis can be seen in Figure 4.12.

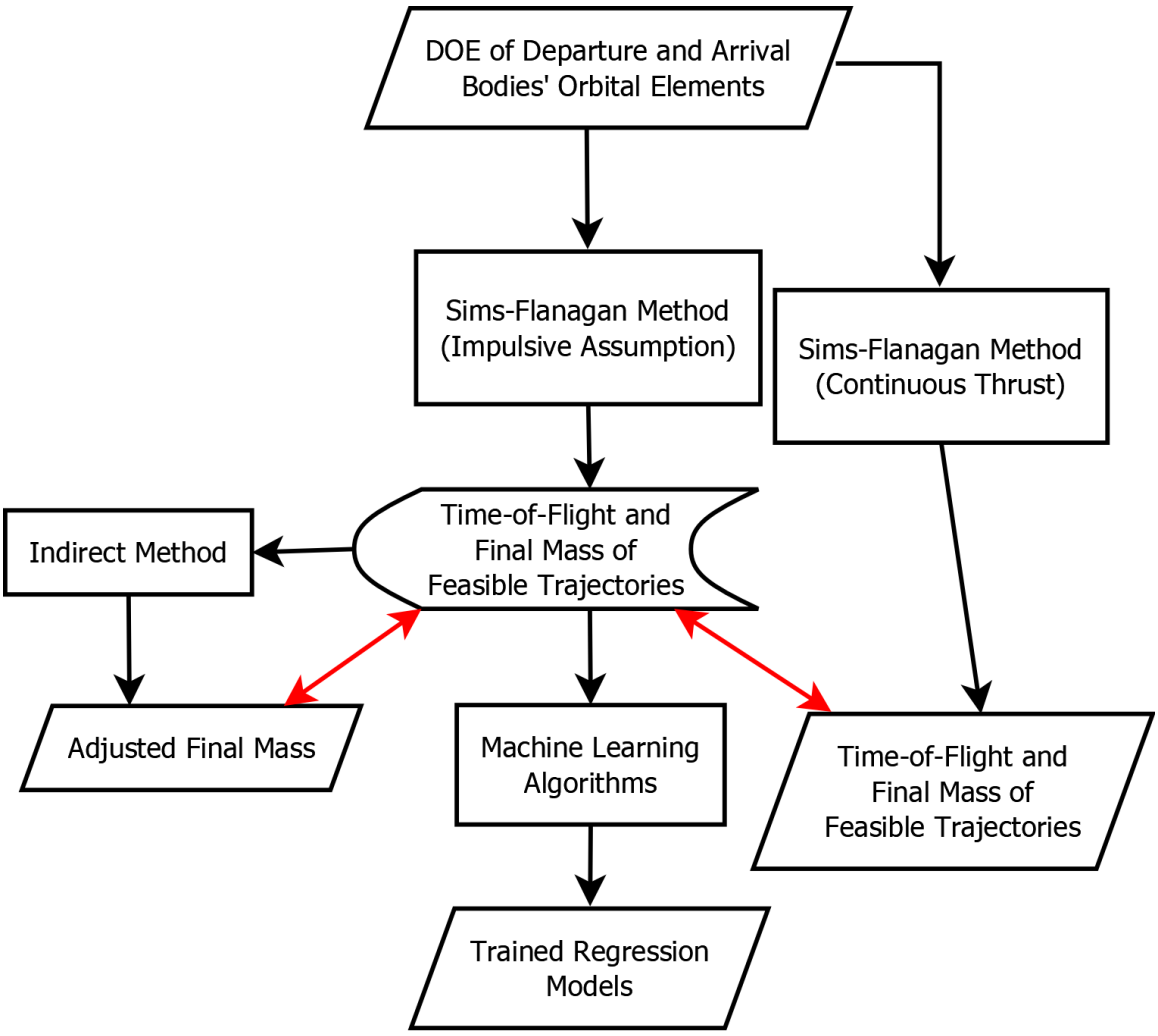


Figure 4.12: Workflow of methodology to create the machine learning estimators. Red arrows indicate comparisons made between models during the experiment for **Research Question 2.**

CHAPTER 5
CONCLUSIONS

5.1 Findings

As a result of the research conducted in this thesis, the ability for machine learning algorithms to predict fuel and time expenditures of generalized low-thrust trajectories has been demonstrated. This is distinct from the results of previous literature in that in this thesis, a much wider variety of orbit transfers is used in the training and validation data sets, and that the orbital flyby problem was investigated, as opposed to the rendezvous problem in previous work. First, **Research Question 1** and **1a** were answered to find a computationally quick way to generate feasible trajectories. It was found that PyKEP's implementation of the Sims-Flanagan method was an effective way to create training data, as opposed to an indirect method or the author's implementation of the Chebyshev polynomial method, and using the impulsive thrust assumption at segment midpoints did not introduce significant error. The inputs for the Sims-Flanagan trajectory solver to create feasible final mass and time-of-flight for transfers came from the Latin Hypercube design of experiments in order to maximize the information gained while minimizing the number of trajectories that need to be found. After the machine learning algorithms had been trained, the resulting errors were in the range of the error observed between different levels of fidelity and optimality in trajectory solvers. Even though the results of previous machine learning predictions of mass expenditure could not be directly compared, a similar gain in performance when compared to the Lambert predictor was seen, suggesting that such a procedure would be applicable for pruning the design space for the best low-thrust transfers. It was also observed that the errors of the machine learning algorithms were not overwhelmingly large compared to the difference between the indirect and direct solvers. For these reasons, the

application of machine learning techniques to a larger design space of transfers was considered successful and **Research Question 2** was answered. The results of this effort would be beneficial to those facing GTOC-class problems. More specifically, multiphase missions involving low thrust propulsion would benefit the most from having an accurate, computationally cheap estimator. Certain global trajectory optimization methods such as Particle Swarm Optimization (PSO), various tree search methods, and genetic algorithms all traditionally require solving many sequences of transfers, and by extension, finding low-thrust trajectories. Having a regressor estimate the fuel and time expenditure for each transfer of each sequence could dramatically decrease the amount of time necessary to run those global trajectory optimization schemes. Complementarily, the regressor is meant to work for a wide variety of two-body problem orbit transfers. This enables its use for not only the asteroid hopping problem, but also the space debris flyby problem, since space debris is not only found in low-inclination orbits like asteroids are almost invariably in. The resultant work is most suited for studies involving the space debris flyby problem, but the methodology may be easily adapted to other orbit regimes. For example, since the training data consists of transfers created using the rendezvous-to-flyby mode for the method, it would be a simple modification to change the mode to a flyby-to-flyby or a rendezvous-rendezvous, or flyby-rendezvous transfers, though this would require creating additional training data. It is expected that the methodology used in this thesis will be applicable to other transfer types.

5.2 Future Work

Future work could be done to improve the efficiency of the indirect solver. As stated previously, indirect solvers can be faster than direct solvers if the initial guess is of high quality. However, in this work, the indirect solver is only given information on the transfer's initial state. By modifying PyGMO's indirect solver, or by using another indirect solver, such as Chebyshev-Picard integration, the full control history output by the direct solver may

be put to use. This would most likely decrease the computation time required. Also, the work in this thesis could be modified to be more in fitting with real-world applications by applying the methodology to the low-Earth orbit regime. This would be done by changing the input data set and instituting certain constraints, such as minimum altitude allowed and non-thrusting when the spacecraft is in Earth's shadow, as well as perturbations, e.g. atmospheric drag, Earth oblateness. The effect of these perturbations have yet to be quantified. Since this thesis concentrates on unperturbed transfers between orbits with vastly differing directions of angular momentum, the choice of heliocentric orbit regime, where high-inclination objects are seldom found, in this very same thesis, may limit the direct practical application of the estimators created. Nonetheless, the process described in this thesis serves as a stepping stone to higher-fidelity machine learning estimators for transfers between dissimilar orbits. Lastly, it would be interesting to see what results a global trajectory optimizer using the techniques as described in this thesis could come up with in a time-limited scenario, where the benefits of such an approach are most apparent. Applications could include GTOC events, where there is little time to implement and run an optimizer, or real-time simulations of pursuit-evasion games between two satellites that attempt to cause and prevent an interception, respectively.

REFERENCES

[1] *Space Debris by the Numbers*. European Space Agency, 2018.

[2] A. Chamberlin, *Discovery Statistics*. NASA Center for Near Earth Object Studies, 2018.

[3] D. Izzo, "1st ACT global trajectory optimisation competition: Problem description and summary of the results," in *Acta Astronautica*, vol. 60, 2007, pp. 731–734.

[4] J. E. Foster, T. Haag, M. Patterson, G. J. Williams, J. S. Slovey, C. Carpenter, H. Kamhawi, S. Malone, and F. Elliot, "The High Power Electric Propulsion (HiPEP) Ion Thruster," in *40th Joint Propulsion Conference*, Fort Lauderdale, FL, 2004.

[5] M. Kim, "Continuous low-thrust trajectory optimization: Techniques and applications," PhD dissertation, Virginia Polytechnic Institute and State University, 2005.

[6] R. R. Bate, D. D. Mueller, and J. E. White, *Fundamentals of Astrodynamics*. Dover Publications, Inc., 1971.

[7] P. R. Patel, "Automating interplanetary trajectory generation for electric propulsion trade studies," PhD dissertation, University of Michigan, 2008.

[8] L. F. Simões, D. Izzo, E. Haasdijk, and A. E. Eiben, "Multi-rendezvous spacecraft trajectory optimization with beam p-aco," in *17th European Conference on Evolutionary Computation in Combinatorial Optimization*, Apr. 2017, pp. 141–156.

[9] K. Alemany, "Design space pruning heuristics and global optimization method for conceptual design of low-thrust asteroid tour missions," PhD dissertation, Georgia Institute of Technology, 2009.

[10] A. Mereta, D. Izzo, and A. Wittig, "Machine Learning of Optimal Low-thrust Transfers between Near-Earth Objects," in *Hybrid Artificial Intelligent Systems*, Cham: Springer International Publishing, 2017, pp. 543–553.

[11] F. Jiang, H. Baoyin, and J. Li, "Practical Techniques for Low-Thrust Trajectory Optimization with Homotopic Approach," in *Journal of Guidance, Control, and Dynamics*, vol. 35, no. 1, 2012, pp. 245–258.

[12] R. S. Park and A. B. Chamberlin, *JPL Small-Body Mission-Design Tool*. JPL Solar System Dynamics Group, 2018.

[13] *Exploring Mars: Porkchop Plot*. NASA, 2005.

[14] D. Izzo, D. Hennes, L. F. Simes, and M. Mrtens, "Designing Complex Interplanetary Trajectories for the Global Trajectory Optimization Competitions," in *Springer Optimization and Its Applications*, vol. 114, Cham: Springer, 2016.

[15] D. Hennes, D. Izzo, and D. Landau, "Fast approximators for optimal low-thrust hops between main belt asteroids," in *IEEE Symposium Series on Computational Intelligence*, 2016.

[16] B. R. Geiger, E. M. Schmidt, and J. F. Horn, "Use of Neural Network Approximation in Multiple-Unmanned Aerial Vehicle Trajectory Optimization," in *AIAA Guidance, Navigation, and Control Conference*, 2009.

[17] A. Ohndorf, "Multiphase low-thrust trajectory optimization using evolutionary neurocontrol," PhD dissertation, Delft University of Technology, 2016.

[18] J. T. Olympio, "Optimal Control Problem for Low-Thrust Multiple Asteroid Tour Missions," in *Journal of Guidance, Control, and Dynamics*, vol. 34, no. 6, 2011, pp. 1709–1719.

[19] C. H. Yam and D. Izzo, "Towards a High Fidelity Direct Transcription Method for Optimisation of Low-Thrust Trajectories," in *International Conference on Astrodynamics Tools and Techniques - ICATT*, 2010.

[20] J. A. Sims and S. N. Flanagan, "Preliminary Design of Low-Thrust Interplanetary Missions," in *American Astronomical Society*, Washington, D.C., 1999.

[21] F. Faroo and I. Ross, "Direct Trajectory Optimization by a Chebyshev Pseudospectral Method," in *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, 2002, pp. 160–166.

[22] T. Antony and M. J. Grant, "Quasilinear chebyshev-picard iteration method for indirect trajectory optimization," Jan. 2019.

[23] R. Bonalli, B. Hriss, and E. Trlat, "Analytical Initialization of a Continuation-Based Indirect Method for Optimal Control of Endo-Atmospheric Launch Vehicle Systems," in *IFAC-PapersOnLine*, 2017.

[24] D. Izzo, C. Sprague, and D. Tailor, "Machine learning and evolutionary techniques in interplanetary trajectory design," in *CoRR*, 2018.

[25] *Near Earth Asteroids (NEAs): A Chronology of Milestones - Page 2*. International Astronomical Union, 2013.

[26] M. Ćuk, B. J. Gladman, and D. Nesvorný, "Hungaria asteroid family as the source of aubrite meteorites,", vol. 239, pp. 154–159, 2014. arXiv: 1406.0825 [astro-ph.EP].

[27] J. Verrelst, J. Muoz, L. Alonso, J. Delegido, J. P. Rivera, G. Camps-Valls, and J. Moreno, "Machine learning regression algorithms for biophysical parameter retrieval: Opportunities for Sentinel-2 and -3," in *Remote Sens. Environ.*, vol. 118, pp. 127–139.

[28] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[29] G. K. F. Tso and K. K. W. Yau, "Predicting electricity energy consumption: A comparison of regression analysis, decision tree and neural networks," in *Energy*, vol. 32, no. 9, 2007, pp. 1761–1768.

[30] G. De'ath, "Boosted Trees for Ecological Modeling and Prediction," in *Ecology*, vol. 88, no. 1, 2007, pp. 243–251.

[31] S. Nawar and A. M. Mouazen, "Comparison between random forests, artificial neural networks and gradient boosted machines methods of on-line vis-nir spectroscopy measurements of soil total nitrogen and total carbon," 2017.

[32] A. J. Smola and B. Schlkopf, "A tutorial on support vector regression," STATISTICS and COMPUTING, Tech. Rep., 2003.

[33] D. A. Herman, "NASA's Evolutionary Xenon Thruster (NEXT) Project Qualification Propellant Throughput Milestone: Performance, Erosion, and Thruster Service Life Prediction After 450 kg," in *57th Joint Army-Navy-NASA-Air Force (JANNAF) Propulsion Meeting*, Nov. 2010.

[34] *Orbital Elements*. Swinburne University of Technology.

[35] P. R. Patel, A. D. Gallimore, T. H. Zurbuchen, and D. J. Scheeres, "An Algorithm for Generating Feasible Low Thrust Interplanetary Trajectories," in *31st International Electric Propulsion Conference*, 2009.

[36] E. W. Weisstein, *Chebyshev Polynomial of the First Kind*. Wolfram Mathworld, 2008.

[37] D. Izzo, "Pygmo and pykep: Open source tools for massively parallel optimization in astrodynamics (the case of interplanetary trajectory optimization)," Jan. 2012.

[38] J. D. Aziz, J. S. Parker, D. J. Scheeres, and J. A. Englander, "Low-thrust many-revolution trajectory optimization via differential dynamic programming and a sundman transformation," *The Journal of the Astronautical Sciences*, vol. 65, no. 2, pp. 205–228, 2018.

[39] J. Sims, P. A. Finlayson, E. A. Rinderle, M. A. Vavrina, and T. D. Kowalkowski, "Implementation of a low-thrust trajectory optimization algorithm for preliminary design," Aug. 2006.

[40] D. Wales and J. Doye, "Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms," *The Journal of Physical Chemistry A*, vol. 101, Apr. 1998.

[41] D. Mavris, *Design-of-Experiments For Practical Applications in Modeling, Simulation, and Analysis: Introduction to Response Surface Methods*. Aerospace Systems Design Laboratory, Georgia Institute of Technology.

[42] Dario Izzo, *Esa/pykep: Major update.* Nov. 2017.

[43] P. E. Gill, W. Murray, M. A. Saunders, and E. Wong, "User's guide for SNOPT 7.7: Software for large-scale nonlinear programming," Department of Mathematics, University of California, San Diego, La Jolla, CA, Center for Computational Mathematics Report CCoM 18-1, 2018.

[44] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Rev.*, vol. 47, pp. 99–131, 2005.

[45] J. M. Little and E. Y. Choueiri, "Electric propulsion system scaling for asteroid capture-and-return missions," in *49th AIAA/ASME/SAE/ASEE Joint Propulsion Conference, Joint Propulsion Conferences*, 2013.

[46] D. V. Byrnes, T. T. McConaghy, and J. M. Longuski, "Analysis of various two synodic period earth-mars cycler trajectories," in *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Aug. 2002.

[47] X. Yue, Y. Yang, and Z. Geng, "Continuous low-thrust time-optimal orbital maneuver," Dec. 2009, pp. 1457–1462.